16th Scandinavian Symposium and Workshops on Algorithm Theory

SWAT 2018, June 18–20, 2018,
Malmö University, Malmö, Sweden

Edited by

David Eppstein

Editors

David Eppstein
Computer Science Department
University of California, Irvine
Irvine, California, USA
eppstein@uci.edu

ACM Classification 2012
Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-068-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-068-2.

Publication date
June, 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.SWAT.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The Scandinavian Symposium and Workshops on Algorithm Theory (SWAT, formerly the Scandinavian Workshop on Algorithm Theory) has been offered every two years beginning in 1988, when it was offered in Halmstaf, Sweden. It alternates with its sister conference, the Algorithms and Data Structures Symposium (WADS), usually held in Canada. This year marks the 16th SWAT, and the fourth time the conference has been in Sweden.

92 regular papers were submitted to the conference; four were withdrawn, and the program committee selected 30 of the remaining 88 papers for presentation at the conference. In addition, the conference program includes three invited talks, whose abstracts are included in the proceedings.

The SWAT conference series is run by a steering committee consisting of Lars Arge (Aarhus University), Magnús M. Halldórsson (Reykjavík University), Andrzej Lingas (Lund University), Jan Arne Telle (University of Bergen), and Esko Ukkonen (University of Helsinki). This year's conference is organized by Jesper Larsson and Bengt J. Nilsson (both of Malmö University).

The program committee consisted of Mikkel Abrahamsen (University of Copenhagen), Joan Boyar (University of Southern Denmark), Jingsen Chen (Luleå University of Technology), Devdatt Dubhashi (Chalmers University of Technology), David Eppstein (chair; University of California, Irvine), Zachary Friggstad (University of Alberta), Travis Gagie (Diego Portales University), Serge Gaspers (University of New South Wales), Iyad Kanj (DePaul University), Viggo Kann (KTH Royal Institute of Technology), Tsvi Kopelowitz (University of Waterloo), Christian Knauer (University of Bayreuth), Irina Kostitsyna (Eindhoven University of Technology), Shi Li (University at Buffalo), Daniel Lokshtanov (University of Bergen), Matthias Mnich (Maastricht University and Rheinische Friedrich-Wilhelms-Universität Bonn), Sang-il Oum (Korea Advanced Institute of Science and Technology), Daniel Paulusma (Durham University), Marcin Pilipczuk (University of Warsaw), Benjamin Raichel (University of Texas at Dallas), Marcel Roeloffzen (National Institute of Informatics), Barna Saha (University of Massachusetts Amherst), Jukka Suomela (Aalto University), and Haitao Wang (Utah State University).

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sampling-Based Motion Planning: From Intelligent CAD to Crowd Simulation to Protein Folding

Nancy M. Amato
Department of Computer Science and Engineering, Texas A&M University
College Station, Texas, USA
amato@tamu.edu

Abstract

Motion planning has application in robotics, animation, virtual prototyping and training, and even for seemingly unrelated tasks such as evaluating architectural plans or simulating protein folding. Surprisingly, sampling-based planning methods have proven effective on problems from all these domains. In this talk, we provide an overview of sampling-based planning and describe some variants developed in our group, including strategies suited for manipulation planning and for user interaction. For virtual prototyping, we show that in some cases a hybrid system incorporating both an automatic planner and haptic user input leads to superior results. For crowd simulation, we describe techniques for evacuation planning and for evaluating architectural designs. Finally, we describe our application of sampling-based motion planners to simulate molecular motions, such as protein and RNA folding.

2012 ACM Subject Classification Computing methodologies → Robotic planning

Keywords and phrases motion planning, probabilistic roadmap

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.1

Category Invited Talk

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Optimizing Society? Ensuring Fairness in Automated Decision-Making

Sorelle Friedler
Department of Computer Science, Haverford College
Haverford, Pennsylvania, USA
sfriedle@haverford.edu

Abstract

Algorithms are increasingly used to make high-stakes decisions about people; who goes to jail, what neighborhoods police deploy to, and who should be hired for a job. But if we want these decisions to be fair, this means we must take societal notions of fairness and express them using the language of math. What is a fair decision and how can it be guaranteed?

In this talk, we'll discuss recent work from the new and growing field of Fairness, Accountability, and Transparency. We will examine technical definitions of fairness and non-discrimination that have been proposed and their societal counterparts. We'll also discuss methods for ensuring that algorithms are making decisions as desired, from methods to audit black-box algorithms to white-box interpretability techniques. This important field necessitates societally informed and mathematically rigorous work; we'll discuss open problems in this light.

2012 ACM Subject Classification Security and privacy → Social aspects of security and privacy

Keywords and phrases algorithmic fairness

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.2

Category Invited Talk

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Robustness Meets Algorithms

Ankur Moitra
Department of Mathematics, MIT
Cambridge, Massachusetts, USA
moitra@mit.edu

Abstract

In every corner of machine learning and statistics, there is a need for estimators that work not just in an idealized model but even when their assumptions are violated. Unfortunately in high-dimensions, being provably robust and efficiently computable are often at odds with each other. In this talk, we give the first efficient algorithm for estimating the parameters of a high-dimensional Gaussian which is able to tolerate a constant fraction of corruptions that is independent of the dimension. Prior to our work, all known estimators either needed time exponential in the dimension to compute, or could tolerate only an inverse polynomial fraction of corruptions. Not only does our algorithm bridge the gap between robustness and algorithms, it turns out to be highly practical in a variety of settings.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms, Computing methodologies → Machine learning algorithms

Keywords and phrases robust estimators, machine learning algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.3

Category Invited Talk

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein; Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Economical Delone Sets for Approximating Convex Bodies

Ahmed Abdelkader
Department of Computer Science
University of Maryland, College Park MD, USA
akader@cs.umd.edu
https://orcid.org/0000-0002-6749-1807

David M. Mount
Department of Computer Science and Institute of Advanced Computer Studies
University of Maryland, College Park MD, USA
mount@cs.umd.edu

Abstract

Convex bodies are ubiquitous in computational geometry and optimization theory. The high combinatorial complexity of multidimensional convex polytopes has motivated the development of algorithms and data structures for approximate representations. This paper demonstrates an intriguing connection between convex approximation and the classical concept of Delone sets from the theory of metric spaces. It shows that with the help of a classical structure from convexity theory, called a Macbeath region, it is possible to construct an ε-approximation of any convex body as the union of $O(1/\varepsilon^{(d-1)/2})$ ellipsoids, where the center points of these ellipsoids form a Delone set in the Hilbert metric associated with the convex body. Furthermore, a hierarchy of such approximations yields a data structure that answers ε-approximate polytope membership queries in $O(\log(1/\varepsilon))$ time. This matches the best asymptotic results for this problem, by a data structure that both is simpler and arguably more elegant.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Approximate polytope membership, Macbeath regions, Delone sets, Hilbert geometry

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.4

Funding Research supported by NSF grant CCF–1618866.

1 Introduction

We consider the following fundamental query problem. Let K denote a bounded convex polytope in $\mathbb{R}^d$, presented as the intersection of n halfspaces. The objective is to preprocess K so that, given any query point $q \in \mathbb{R}^d$, it is possible to determine efficiently whether q lies in K. Throughout, we assume that d is a fixed constant and K is full-dimensional.

Polytope membership is equivalent in the dual setting to answering halfspace emptiness queries for a set of n points in $\mathbb{R}^d$. In dimensions higher than three, the fastest exact data structure with near-linear space has a query time of roughly $O\big(n^{1-1/\lfloor d/2 \rfloor}\big)$ [29], which is unacceptably high for many applications. Hence, we consider an approximate setting.

Let ε be a positive real parameter, and let $\mathrm{diam}(K)$ denote K's diameter. Given a query point $q \in \mathbb{R}^d$, an *ε-approximate polytope membership query* returns a positive result if $q \in K$, a negative result if the distance from q to its closest point in K is greater than $\varepsilon \cdot \mathrm{diam}(K)$, and it may return either result otherwise.

Polytope membership queries, both exact and approximate, arise in many application areas, such as linear programming and ray-shooting queries [15, 28, 30, 33], nearest neighbor searching and the computation of extreme points [1, 16, 18], collision detection [21], and machine learning [14].

Dudley [20] showed that, for any convex body K in $\mathbb{R}^d$, it is possible to construct an ε-approximating polytope P with $O(1/\varepsilon^{(d-1)/2})$ facets. This bound is asymptotically tight, and is achieved when K is a Euclidean ball. This construction implies a (trivial) data structure for approximate polytope membership problem with space and query time $O(1/\varepsilon^{(d-1)/2})$. It follows from the work of Bentley *et al.* [11] that there is a simple grid-based solution, that answers queries in constant time using space $O(1/\varepsilon^{d-1})$. Arya *et al.* [2, 3] present algorithms that achieve a tradeoff between these two extremes, but their data structure provides no improvement over the storage in [11] when the query time is polylogarithmic.

A space-optimal solution for the case of polylogarithmic query time was presented in [7]. It achieves query time $O(\log \frac{1}{\varepsilon})$ with storage $O(1/\varepsilon^{(d-1)/2})$. This paper achieves its efficiency by abandoning the grid- and quadtree-based approaches in favor of an approach based on ellipsoids and a classical structure from convexity theory called a *Macbeath region* [27].

The approach presented in [7] is based on constructing a collection of nested eroded bodies within K and covering the boundaries of these eroded bodies with ellipsoids that are based on Macbeath regions. Queries are answered by shooting rays from a central point in the polytope towards the boundary of K, and tracking an ellipsoid at each level that is intersected by the ray. While it is asymptotically optimal, the data structure and its analysis are complicated by various elements that are artifacts of this ray shooting approach.

In this paper, we present a simpler and more intuitive approach with the same asymptotic complexity as the one in [7]. The key idea is to place the Macbeath regions based on *Delone sets*. A Delone set is a concept from the study of metric spaces. It consists of a set of points that have nice packing and covering properties with respect to the metric balls. Our main result is that any maximal set of disjoint shrunken Macbeath regions defines a Delone set with respect to the Hilbert metric induced on a suitable expansion of the convex body. This observation leads to a simple DAG structure for membership queries. The DAG structure arises from a hierarchy of Delone sets obtained by layering a sequence of expansions of the body. Our results uncover a natural connection between the classical concepts of Delone sets from the theory of metric spaces and Macbeath regions and the Hilbert geometry from the theory of convexity.

2 Preliminaries

In this section we present a number of basic definitions and results, which will be used throughout the paper. We consider the real d-dimensional space, $\mathbb{R}^d$, where d is a fixed constant. Let O denote the origin of $\mathbb{R}^d$. Given a vector $v \in \mathbb{R}^d$, let $\|v\|$ denote its Euclidean length, and let $\langle \cdot, \cdot \rangle$ denote the standard inner product. Given two points $p, q \in \mathbb{R}^d$, the Euclidean distance between them is $\|p - q\|$. For $q \in \mathbb{R}^d$ and $r > 0$, let $B(q, r)$ denote the Euclidean ball of radius r centered at q, and let $B(r) = B(O, r)$.

Let K be a convex body in $\mathbb{R}^d$, represented as the intersection of m closed halfspaces $H_i = \{x \in \mathbb{R}^d : \langle x, v_i \rangle \leq a_i\}$, where a_i is a nonnegative real and $v_i \in \mathbb{R}^d$. The bounding hyperplane for H_i is orthogonal to v_i and lies at distance $a_i/\|v_i\|$ from the origin. The boundary of K will be denoted by ∂K. For $0 < \kappa \leq 1$, we say that K is in *κ-canonical form* if $B(\kappa/2) \subseteq K \subseteq B(1/2)$. Clearly, such a body has a diameter between κ and 1.

It is well known that in $O(m)$ time it is possible to compute a non-singular affine transformation T such that $T(K)$ is in $(1/d)$-canonical form [6, 23]. Further, if a convex body P is within Hausdorff distance ε of $T(K)$, then $T^{-1}(P)$ is within Hausdorff distance at most $d\varepsilon$ of K. (Indeed, this transformation is useful, since the resulting approximation is directionally sensitive, being more accurate along directions where K is skinnier.) Therefore, for the sake of approximation with respect to Hausdorff distance, we may assume that K has been mapped to canonical form, and ε is scaled by a factor of $1/d$. Because we assume that d is a constant, this transformation will only affect the constant factors in our analysis.

A number of our constructions involve perturbing the body K by means of *expansion*, but the exact nature of the expansion is flexible in the following sense. Given $\delta > 0$, let K_δ denote any convex body containing K such that the Hausdorff distance between ∂K and ∂K_δ is $\Theta(\delta \cdot \mathrm{diam}(K))$. For example, if K is in canonical form, K_δ could result as the Minkowski sum of K with another convex body of diameter δ or from a uniform scaling about the origin by δ. Because reducing the approximation parameter by a constant factor affects only the constant factors in our complexity bounds, the use of an appropriate K_δ instead of closely related notions of approximation, like the two just mentioned, will not affect our asymptotic bounds. Given $\delta > 0$, we perturb each H_i to obtain

$$H_{i,\delta} \;=\; \{x \in \mathbb{R}^d : \langle x, \vec{v}_i\rangle \le a_i + \delta)\}.$$

The associated bounding hyperplane is parallel to that of H_i and translated away from the origin by a distance of $\delta/\|v_i\|$. With that, we define K_δ as the convex polytope $\bigcap_{i=1}^n H_{i,\delta}$. To ensure the required bound on the Hausdorff error, we require that $c_1\delta \le \|v_i\| \le c_2$ for all i, where c_1 and c_2 are nonnegative reals. The following argument shows that this condition suffices. If $c_1\delta \le \|v_i\| \le c_2$, then each bounding halfspace of K is translated away from the origin by a distance of $\delta/\|v_i\| \ge \delta/c_2$, which establishes the lower bound on the Hausdorff distance. Also, each bounding halfspace is translated by a distance of $\delta/\|v_i\| \le 1/c_1$. Since K, being in canonical form, is nested between balls of radius $\kappa/2$ and $1/2$, this translation of the halfspace is equivalent to a scaling about the origin by a factor of at most $2/c_1\kappa$, which maps each point of K away from the origin by a distance of at most $(2/c_1\kappa)/2 = 1/c_1\kappa$. This establishes the upper bound on the Hausdorff distance.

2.1 Macbeath regions

Our algorithms and data structures will involve packings and coverings by ellipsoids, which will possess the essential properties of Delone sets. These ellipsoids are based on a classical concept from convexity theory, called *Macbeath regions*, which were described first by A. M. Macbeath in a paper on the existence of certain lattice points in a convex body [27]. They have found uses in diverse areas (see, e.g., Bárány's survey [9]).

Given a convex body K, a point $x \in K$, and a real parameter $\lambda \ge 0$, the λ-*scaled Macbeath region* at x, denoted $M_K^\lambda(x)$, is defined to be

$$x + \lambda((K - x) \cap (x - K)).$$

When $\lambda = 1$, it is easy to verify that $M_K^1(x)$ is the intersection of K and the reflection of K around x (see Fig. 1a), and hence it is centrally symmetric about x. $M_K^\lambda(x)$ is a scaled copy of $M_K^1(x)$ by the factor λ about x. We refer to x and λ as the *center* and *scaling factor* of $M_K^\lambda(x)$, respectively. To simplify the notation, when K is clear from the context, we often omit explicit reference in the subscript and use $M^\lambda(x)$ in place of $M_K^\lambda(x)$. When $\lambda < 1$, we

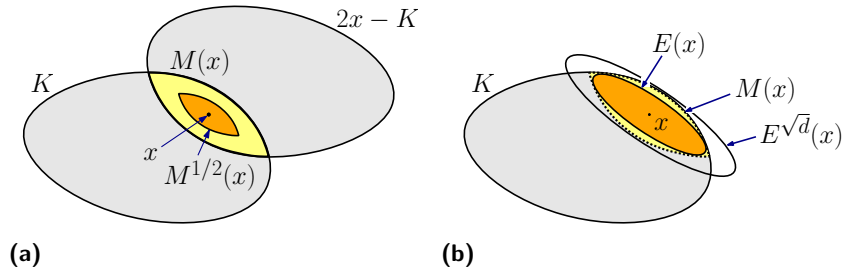


Figure 1 (a) Macbeath regions and (b) Macbeath ellipsoids.

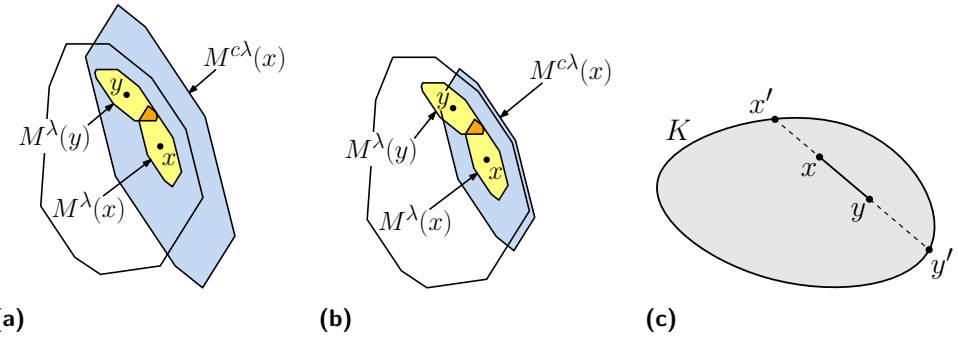


Figure 2 (a)-(b) Expansion-containment per Lemma 1. (c) The Hilbert metric.

say $M^\lambda(x)$ is *shrunken*. When $\lambda = 1$, $M^1(x)$ is *unscaled* and we drop the superscript. Recall that if C^λ is a uniform λ-factor scaling of any bounded, full-dimensional set $C \subset \mathbb{R}^d$, then $\mathrm{vol}(C^\lambda) = \lambda^d \cdot \mathrm{vol}(C)$.

An important property of Macbeath regions, which we call *expansion-containment*, is that if two shrunken Macbeath regions overlap, then an appropriate expansion of one contains the other (see Fig. 2a). The following is a generalization of results of Ewald, Rogers and Larman [22] and Brönnimann, Chazelle, and Pach [13]. Our generalization allows the shrinking factor λ to be adjusted, and shows how to adjust the expansion factor β of the first body to cover an α-scaling of the second body, e.g., the center point only (see Fig. 2b).

▶ **Lemma 1.** *Let $K \subset \mathbb{R}^d$ be a convex body and let $0 < \lambda < 1$. If $x, y \in K$ such that $M^\lambda(x) \cap M^\lambda(y) \neq \emptyset$, then for any $\alpha \geq 0$ and $\beta = \frac{2+\alpha(1+\lambda)}{1-\lambda}$, $M^{\alpha\lambda}(y) \subseteq M^{\beta\lambda}(x)$ (see Fig. 2).*

2.2 Delone sets and the Hilbert metric

An important concept in the context of metric spaces involves coverings and packings by metric balls [19]. Given a metric f over $\mathbb{X}$, a point $x \in \mathbb{X}$, and real $r > 0$, define the ball $B_f(x, r) = \{y \in \mathbb{X} : f(x, y) \leq r\}$. For $\varepsilon, \varepsilon_p, \varepsilon_c > 0$, a set $X \subseteq \mathbb{X}$ is an:
ε-packing: If the balls of radius $\varepsilon/2$ centered at every point of X do not intersect.
ε-covering: If every point of $\mathbb{X}$ is within distance ε of some point of X.
$(\varepsilon_p, \varepsilon_c)$-Delone Set: If X is an ε_p-packing and an ε_c-covering.

Delone sets have been used in the design of data structures for answering geometric proximity queries in metric spaces through the use of hierarchies of nets, such as navigating nets [26], net trees [24], and cover trees [12].

In order to view a collection of Macbeath regions as a Delone set, it will be useful to introduce an underlying metric. The Hilbert metric [25] was introduced over a century ago

by David Hilbert as a generalization of the Cayley-Klein model of hyperbolic geometry. A *Hilbert geometry* (K, f_K) consists of a convex domain K in $\mathbb{R}^d$ with the Hilbert distance f_K. For any pair of distinct points $x, y \in K$, the line passing through them meets ∂K at two points x' and y'. We label these points so that they appear in the order $\langle x', x, y, y' \rangle$ along this line (see Fig. 2c). The Hilbert distance f_K is defined as

$$f_K(x, y) \;=\; \frac{1}{2} \ln \left(\frac{\|x' - y\|}{\|x' - x\|} \frac{\|x - y'\|}{\|y - y'\|} \right).$$

When K is not bounded and either x' or y' is at infinity, the corresponding ratio is taken to be 1. To get some intuition, observe that if x is fixed and y moves along a ray starting at x towards ∂K, $f_K(x, y)$ varies from 0 to ∞.

Hilbert geometries have a number of interesting properties; see the survey by Papadopoulos and Troyanov [32] and the multimedia contribution by Nielsen and Shao [31]. First, f_K can be shown to be a metric. Second, it is invariant under projective transformations.[1] Finally, when K is a unit ball in $\mathbb{R}^d$, the Hilbert distance is equal (up to a constant factor) to the distance between points in the Cayley-Klein model of hyperbolic geometry.

Given a point $x \in K$ and $r > 0$, let $B_H(x, r)$ denote the ball of radius r about x in the Hilbert metric. The following lemma shows that a shrunken Macbeath region is nested between two Hilbert balls whose radii differ by a constant factor (depending on the scaling factor). Thus, up to constant factors in scaling, Macbeath regions and their associated ellipsoids can act as proxies to metric balls in Hilbert space. This nesting was observed by Vernicos and Walsh [34] (for the conventional case of $\lambda = 1/5$), and we present the straightforward generalization to other scale factors. For example, with $\lambda = 1/5$, we have $B_H(x, 0.09) \subseteq M^{1/5}(x) \subseteq B_H(x, 0.21)$ for all $x \in K$.

▶ **Lemma 2.** *Given a convex body $K \subset \mathbb{R}^d$, for all $x \in K$ and any $0 \leq \lambda < 1$,*

$$B_H\big(x, \frac{1}{2} \ln (1 + \lambda)\big) \;\subseteq\; M^{\lambda}(x) \;\subseteq\; B_H\left(x, \frac{1}{2} \ln \frac{1 + \lambda}{1 - \lambda}\right).$$

3 Macbeath regions as Delone sets

Lemma 2 justifies using Macbeath regions as Delone sets. Given a point $x \in K$ and $\delta > 0$, define $M_\delta(x)$ to be the (unscaled) Macbeath region with respect to K_δ, that is, $M_\delta(x) = M_{K_\delta}(x)$. Towards our goal of using Delone sets for approximating convex bodies, we study the behavior of overlapping Macbeath regions at different scales of approximation and establish a bound on the size of such Delone sets. In particular, we consider maximal sets of disjoint shrunken Macbeath regions $M_\delta^\lambda(x)$ defined with respect to K_δ, such that the centers x lie within K; let X_δ denote such a set of centers. The two scale factors used to define the Delone set will be denoted by (λ_p, λ_c), where we assume $0 < \lambda_p < \lambda_c < 1$ are constants. Define $M'_\delta(x) = M_\delta^{\lambda_c}(x)$ and $M''_\delta(x) = M_\delta^{\lambda_p}(x)$.

3.1 Varying the scale

A crucial property of metric balls is how they adapt to changing the resolution at which the domain in question is being modeled. We show that Macbeath regions enjoy a similar property.

[1] This follows from the fact that the argument to the logarithm function is the *cross ratio* of the points (x', x, y, y'), and it is well known that cross ratios are preserved under projective transformations.

▶ **Lemma 3.** *Given a convex body $K \subset \mathbb{R}^d$ and $\lambda, \delta, \varepsilon \geq 0$, for all $x \in K$,*

$$M^{\lambda}_{K_\delta}(x) \;\subseteq\; M^{\lambda}_{K_{(1+\varepsilon)\delta}}(x) \;\subseteq\; M^{(1+\varepsilon)\lambda}_{K_\delta}(x).$$

Proof. The first inclusion is a simple consequence of the fact that enlarging the body can only enlarge the Macbeath regions. To see the second inclusion, it will simplify the notation to translate space by $-x$ so that x now coincides with the origin. Thus, $M_K(x) = K \cap -K$. Recalling our representation from Section 2, we can express K as the intersection of a set of halfspaces $H_i = \{y : \langle y, v_i\rangle \leq a_i\}$. (The translation affects the value of a_i, but not the approximation, because $x \in K$, $a_i \geq 0$.) We can express $M_K(x)$ as the intersection of a set of slabs $\Sigma_i = H_i \cap -H_i$, where each slab is centered about the origin. $M_{K_\delta}(x)$ can be similarly expressed as the intersection of slabs $\Sigma_{i,\delta} = H_{i,\delta} \cap -H_{i,\delta}$, where the defining inequality is $\langle y, v_i\rangle \leq a_i + \delta$. This applies analogously to $M_{K_{(1+\varepsilon)\delta}}(x)$, where the defining inequality is $\langle y, v_i\rangle \leq a_i + (1+\varepsilon)\delta$. Since $a_i \geq 0$, we have $a_i + (1+\varepsilon)\delta \leq (1+\varepsilon)(a_i + \delta)$, which implies that $\Sigma_{i,(1+\varepsilon)\delta} \subseteq (1+\varepsilon)\Sigma_{i,\delta}$. Thus, we have

$$M_{K_{(1+\varepsilon)\delta}}(x) \;=\; \bigcap\nolimits_{i=1}^{m} \Sigma_{i,(1+\varepsilon)\delta} \;\subseteq\; \bigcap\nolimits_{i=1}^{m} (1+\varepsilon)\Sigma_{i,\delta} \;=\; M^{(1+\varepsilon)}_{K_\delta}(x).$$

The lemma now follows by applying a scaling factor of λ to both sides. ◀

As we refine the approximation by using smaller values of δ, it is important to bound the number of Macbeath regions at higher resolution that overlap any given Macbeath region at a lower resolution. Our bound is based on a simple packing argument. We will show that the shrunken Macbeath regions $M''_\delta(y)$ that overlap a fixed shrunken Macbeath region at a coarser level of approximation $M'_{s\delta}(x)$, with $s \geq 1$, lie within a suitable constant-factor expansion of $M'_{s\delta}(x)$. Let $Y_{\delta,s}(x)$ denote the set of points y such that $M''_\delta(y)$ are pairwise disjoint and overlap $M'_{s\delta}(x)$. Since these shrunken Macbeath regions are pairwise disjoint, we can bound their number by bounding the ratio of volumes of $M'_{s\delta}(x)$ and $M''_\delta(y)$.

As an immediate corollary of the second inclusion of Lemma 3 we have $\mathrm{vol}(M^{\lambda}_{\delta}(x)) \geq \mathrm{vol}(M^{\lambda}_{s\delta}(x))/s^d$. This allows us to establish an upper bound on the growth rate in the number of Macbeath regions when refining to smaller scales.

▶ **Lemma 4.** *Given a convex body $K \subset \mathbb{R}^d$ and $x \in K$. Then, for constants $\delta \geq 0$, $s \geq 1$ and $Y_{\delta,s}(x)$ as defined above, $|Y_{\delta,s}(x)| = O(1)$.*

Proof. By the first inclusion of Lemma 3, $M'_\delta(y) \subseteq M'_{s\delta}(y)$, and we have $M'_{s\delta}(x) \cap M'_{s\delta}(y) \neq \emptyset$. Next, by applying Lemma 1 (with the roles of x and y swapped) we obtain $M'_{s\delta}(x) = M^{\lambda_c}_{s\delta}(x) \subseteq M^{\beta\lambda_c}_{s\delta}(y)$, with $\alpha = 1$ and $\beta = (3+\lambda_c)/(1-\lambda_c)$.

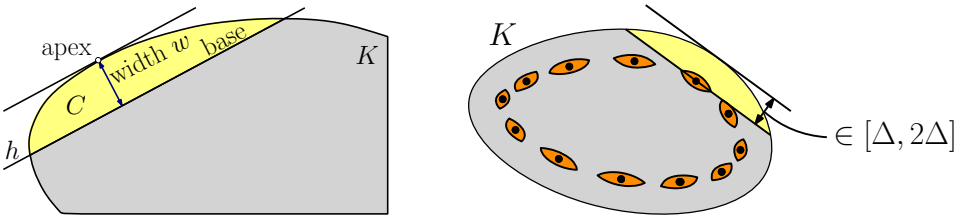
By definition of X_δ the shrunken Macbeath regions $M''_\delta(y)$ are pairwise disjoint, and so it suffices to bound their volumes with respect to that of $M'_{s\delta}(x)$ to obtain a bound on $|Y_{\delta,s}(x)|$. Applying the corollary to Lemma 3 and scaling, we obtain

$$\mathrm{vol}(M''_\delta(y)) \;\geq\; \frac{1}{s^d}\mathrm{vol}(M''_{s\delta}(y)) \;=\; \left(\frac{\lambda_p}{\beta\lambda_c s}\right)^d \mathrm{vol}(M^{\beta\lambda_c}_{s\delta}(y)) \;\geq\; \left(\frac{\lambda_p}{\beta\lambda_c s}\right)^d \mathrm{vol}(M'_{s\delta}(x)).$$

Thus, by a packing argument the number of children is at most $\left(\frac{\beta\lambda_c s}{\lambda_p}\right)^d = O(1)$. ◀

3.2 Size bound

We bound the cardinality of a maximal set of disjoint shrunken Macbeath regions $M^{\lambda}_{\delta}(x)$ defined with respect to K_δ, such that the centers x lie within K; let X_δ denote such a set of centers. This is facilitated by associating each center x with a *cap* of K, where a cap C is



(a) **(b)**

Figure 3 (a) Cap concepts and (b) the economical cap cover.

defined as the nonempty intersection of the convex body K with a halfspace (see Fig. 3a). Letting h denote the hyperplane bounding this halfspace, the *base* of C is defined as $h \cap K$. The *apex* of C is any point in the cap such that the supporting hyperplane of K at this point is parallel to h. The *width* of C is the distance between h and this supporting hyperplane. Of particular interest is a cap of minimum volume that contains x, which may not be unique. A simple variational argument shows that x is the centroid of the base of this cap [22].

As each Macbeath region is associated with a cap, we can obtain the desired bound by bounding the number of associated caps. We achieve this by appealing to the so-called *economical cap covers* [10]. The following lemma is a straightforward adaptation of the width-based economical cap cover per Lemma 3.2 of [6].

▶ **Lemma 5.** *Let $K \subset \mathbb{R}^d$ be a convex body in κ-canonical form. Let $0 < \lambda \leq 1/5$ be any fixed constant, and let $\Delta \leq \kappa/12$ be a real parameter. Let $\mathcal{C}$ be a set of caps, whose widths lie between Δ and 2Δ, such that the Macbeath regions $M_K^\lambda(x)$ centered at the centroids x of the bases of these caps are disjoint. Then $|\mathcal{C}| = O(1/\Delta^{(d-1)/2})$ (see Fig. 3a(b)).*

This leads to the following bound on the number of points in X_δ.

▶ **Lemma 6.** *Let $K \subset \mathbb{R}^d$ be a convex body in κ-canonical form, and let X_δ as defined above for some $\delta > 0$ and $0 < \lambda \leq 1/5$. Then, $|X_\delta| = O(1/\delta^{(d-1)/2})$.*

Proof. In order to apply Lemma 5 we will partition the points of X_δ according to the widths of their minimum-volume caps. For $i \geq 0$, define $\Delta_i = c_2 2^i \delta_i$, where c_2 depends on the nature of the the expansion process that yields K_δ. Define $X_{\delta,i}$ to be the subset of points $x \in X_\delta$ such that width of x's minimum cap with respect to K_δ lies within $[\Delta_i, 2\Delta_i]$. By choosing c_2 properly, the Hausdorff distance between K and K_δ is at least $c_2\delta = \Delta_0$, and therefore any cap whose base passes through a point of X_δ has width at least Δ_0. This implies that every point of X_δ lies in some subset $X_{\delta,i}$ for $i \geq 0$.

If a convex body is in κ-canonical form, it follows from a simple geometric argument that for any point x in this body whose minimal cap is of width at least Δ, the body contains a ball of radius $c\Delta$ centered at x, for some constant c (depending on κ and d). If $\Delta_i > \kappa/12$, then $B(x, c\kappa/12) \subseteq K_\delta$ for all $x \in X_{\delta,i}$. It follows that $B(x, c\kappa/12) \subseteq M_\delta(x)$ implying that $\text{vol}(M_\delta^\lambda(x)) \geq \lambda^d \cdot \text{vol}(B(c\kappa/12))$ which is $\Omega(1)$ as c, κ and λ are all constants. By a simple packing argument $|X_{i,j}| = O(1)$. There are at most a constant number of levels for which $\Delta_j > \kappa/12$, and so the overall contribution of these subsets is $O(1)$.

Henceforth, we may assume that $\Delta_j \leq \kappa/12$. Since $\lambda \leq 1/5$, we apply Lemma 5 to obtain the bound $|X_{\delta,i}| = O(1/\Delta_i^{(d-1)/2})$. (There is a minor technicality here. If δ becomes sufficiently large, K_δ may not be in κ-canonical form because its diameter is too large. Because $\delta = O(1)$ and hence $\text{diam}(K_\delta) = O(1)$, we may scale it back into canonical form at

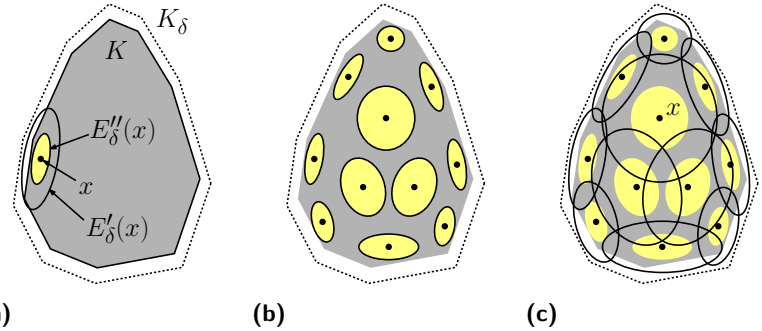


Figure 4 A Delone set for a convex body. (Not drawn to scale.)

the expense of increasing the constant factors hidden in the asymptotic bound.) Thus, up to constant factors, we have

$$|X_\delta| \;=\; \sum_{i\geq 0} |X_{\delta,i}| \;=\; \sum_{i\geq 0} O\left(\frac{1}{\Delta_i}\right)^{\frac{d-1}{2}} \;=\; \sum_{i\geq 0} O\left(\frac{1}{c_2 2^i \delta}\right)^{\frac{d-1}{2}} \;=\; O\left(\left(\frac{1}{\delta}\right)^{\frac{d-1}{2}}\right). \qquad \blacktriangleleft$$

4 Macbeath ellipsoids

For the sake of efficient computation, it will be useful to approximate Macbeath regions by shapes of constant combinatorial complexity. We have opted to use ellipsoids. (Note that bounding boxes [1] could be used instead, and may be preferred in contexts where polytopes are preferred.)

Given a Macbeath region, define its associated *Macbeath ellipsoid* $E_K^\lambda(x)$ to be the maximum-volume ellipsoid contained within $M_K^\lambda(x)$ (see Fig. 1b). Clearly, this ellipsoid is centered at x and $E_K^\lambda(x)$ is an λ-factor scaling of $E_K^1(x)$ about x. It is well known that the maximum-volume ellipsoid contained within a convex body is unique, and Chazelle and Matoušek showed that it can be computed for a convex polytope in time linear in the number of its bounding halfspaces [17]. By John's Theorem (applied in the context of centrally symmetric bodies) it follows that $E_K^\lambda(x) \subseteq M_K^\lambda(x) \subseteq E_K^{\lambda\sqrt{d}}(x)$ [8].

Given a point $x \in K$ and $\delta > 0$, define $M_\delta(x)$ to be the (unscaled) Macbeath region with respect to K_δ (as defined in Section 2), that is, $M_\delta(x) = M_{K_\delta}(x)$. Let $E_\delta(x)$ denote the maximum volume ellipsoid contained within $M_\delta(x)$. As $M_\delta(x)$ is symmetric about x, $E_\delta(x)$ is centered at x. For any $\lambda > 0$, define $M_\delta^\lambda(x)$ and $E_\delta^\lambda(x)$ to be the uniform scalings of $M_\delta(x)$ and $E_\delta(x)$, respectively, about x by a factor of λ. By John's Theorem, we have

$$E_\delta^\lambda(x) \;\subseteq\; M_\delta^\lambda(x) \;\subseteq\; E_\delta^{\lambda\sqrt{d}}(x). \tag{1}$$

Two particular scale factors will be of interest to us. Define $M'_\delta(x) = M_\delta^{1/2}(x)$ and $M''_\delta(x) = M_\delta^{\lambda_0}(x)$, where $\lambda_0 = 1/(4\sqrt{d}+1)$. Similarly, define $E'_\delta(x) = E_\delta^{1/2}(x)$ and $E''_\delta(x) = E_\delta^{\lambda_0}(x)$ (see Fig. 4(a)). Given a fixed δ, let X_δ be any maximal set of points, all lying within K, such that the ellipsoids $E''_\delta(x)$ are pairwise disjoint for all $x \in X_\delta$.

These ellipsoids form a packing of K_δ (see Fig. 4(b)). The following lemma shows that their suitable expansions cover K while being contained within K_δ (see Fig. 4(c)).

▶ **Lemma 7.** *Given a convex body K in $\mathbb{R}^d$ and a set X_δ as defined above for $\delta > 0$,*

$$K \;\subseteq\; \bigcup_{x\in X_\delta} E'_\delta(x) \;\subseteq\; K_\delta.$$

Proof. To establish the first inclusion, consider any point $y \in K$. Because X_δ is maximal, there exists $x \in X_\delta$ such that $E''_\delta(x) \cap E''_\delta(y)$ is nonempty. By containment, $M''_\delta(x) \cap M''_\delta(y)$ is also nonempty. By Lemma 1 (with $\alpha = 0$), it follows that $y \in M^\lambda_\delta(x)$, where

$$\lambda = \frac{2\lambda_0}{1-\lambda_0} = \frac{2/(4\sqrt{d}+1)}{1-1/(4\sqrt{d}+1)} = \frac{2}{4\sqrt{d}} = \frac{1}{2\sqrt{d}}.$$

By applying Eq. (1) (with $\lambda = 1/(2\sqrt{d})$), we have $M^{1/(2\sqrt{d})}_\delta(x) \subseteq E^{1/2}_\delta(x) = E'_\delta(x)$, and therefore $y \in E'_\delta(x)$. Thus, we have shown that an arbitrary point $y \in K$ is contained in the ellipsoid $E'_\delta(x)$ for some $x \in X_\delta$, implying that the union of these ellipsoids covers K. The second inclusion follows from $E'_\delta(x) \subseteq M'_\delta(x) \subseteq M_\delta(x) \subseteq K_\delta$ for any $x \in X_\delta \subseteq K$. ◀

In conclusion, if we treat the scaling factor λ in $E^\lambda(x)$ as a proxy for the radius of a metric ball, we have shown that X_δ is a $(2\lambda_0, 1/2)$-Delone set for K. By Lemma 2 this is also true in the Hilbert metric over K_δ up to a constant factor adjustment in the radii. (Note that the scale of the Hilbert balls does not vary with δ. What varies is the choice of the expanded body K_δ defining the metric.)

By John's Theorem, Macbeath regions and Macbeath ellipsoids differ by a constant scaling factor, both with respect to enclosure and containment. We remark that all the results of the previous two sections hold equally for Macbeath ellipsoids. We omit the straightforward, but tedious, details.

▶ Remark. All results from previous section on scaled Macbeath regions apply to scaled Macbeath ellipsoids subject to appropriate modifications of the constant factors.

5 Approximate polytope membership (APM)

The Macbeath-based Delone sets developed above yield a simple data structure for answering ε-APM queries for a convex body K. We assume that K is represented as the intersection of m halfspaces. We may assume that in $O(m)$ time it has been transformed into κ-canonical form, for $\kappa = 1/d$. Throughout, we will assume that Delone sets are based on the Macbeath ellipsoids $E''_\delta(x)$ for packing and $E'_\delta(x)$ for coverage (defined in Section 4).

Our data structure is based on a hierarchy of Delone sets of exponentially increasing accuracy. Define $\delta_0 = \varepsilon$, and for any integer $i \geq 0$, define $\delta_i = 2^i\delta_0$. Let X_i denote a Delone set for K_{δ_i}. By Lemma 7, we may take X_i to be any maximal set of points within K such that the packing ellipsoids $E''_\delta(x)$ are pairwise disjoint. Let $\ell = \ell_\varepsilon$ be the smallest integer such that $|X_\ell| = 1$. We will show below that $\ell = O(\log 1/\varepsilon)$.

Given the sets $\langle X_0, \ldots, X_\ell \rangle$, we build a rooted, layered DAG structure as follows. The nodes of level i correspond 1–1 with the points of X_i. The leaves reside at level 0 and the root at level ℓ. Each node $x \in X_i$ is associated with two things. The first is its *cell*, denoted $\mathrm{cell}(x)$, which is the covering ellipsoid $E'_\delta(x)$ (the larger hollow ellipsoids shown in Fig. 5). The second, if $i > 0$, is a set of *children*, denoted $\mathrm{ch}(x)$, which consists of the points $y \in X_{i-1}$ such that $\mathrm{cell}(x) \cap \mathrm{cell}(y) \neq \emptyset$.

To answer a query q, we start at the root and iteratively visit any one node $x \in X_i$ at each level of the DAG, such that $q \in \mathrm{cell}(x)$. We know that if q lies within K, such an x must exist by the covering properties of Delone sets, and further at least one of x's children contains q. If q does not lie within any of the children of the current node, the query algorithm terminates and reports (without error) that $q \notin K$. Otherwise the search eventually reaches a node $x \in X_0$ at the leaf level whose cell contains q. Since $\mathrm{cell}(x) \subseteq K_{\delta_0} = K_\varepsilon$, this cell serves as a witness to q's approximate membership within K.

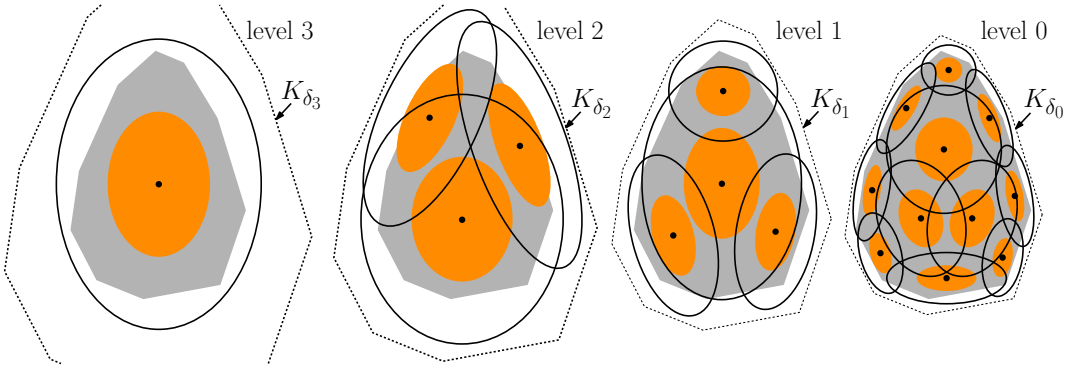


Figure 5 Hierarchy of ellipsoids for answering APM queries.

In order to bound the space and query time, we need to bound the total space used by the data structure and the time to process each node in the search, which is proportional to the number of its children. Building upon Lemmas 4 and 6, we have our main result.

▶ **Theorem 8.** *Given a convex body K and $\varepsilon > 0$, there exists a data structure of space $O(1/\varepsilon^{(d-1)/2})$ that answers ε-approximate polytope membership queries in time $O(\log 1/\varepsilon)$.*

Since the expansion factors δ_i grow exponentially from ε to a suitably large constant, it follows that the height of the tree is logarithmic in $1/\varepsilon$, which is made formal below.

▶ **Lemma 9.** *The DAG structure described above has height $O(\log 1/\varepsilon)$.*

Proof. Let c_2 be an appropriate constant, and let $\ell = \lceil \log_2(2/c_2\varepsilon) \rceil = O(\log 1/\varepsilon)$. Depending the nature of the expanded body K_δ, the constant c_2 can be chosen so the Hausdorff distance between K and K_{δ_ℓ} is at least $c_2\delta_\ell = c_2 2^\ell \varepsilon \geq 2$. Because K is in κ-canonical form, it is contained within a unit ball centered at the origin. Therefore, K_{δ_ℓ} contains a ball of radius two centered at the origin, which implies that the Macbeath ellipsoid $E'_{\delta_\ell}(O)$ (which is scaled by $1/2$) contains the unit ball and so contains K. Thus, (assuming that the origin is added first to the Delone set) level ℓ of the DAG contains a single node. ◀

By Lemma 4, each node has $O(1)$ children and $\delta_i = 2^i\delta_0 = 2^i\varepsilon$, we obtain the following space bound by summing $|X_i|$ for $0 \leq i \leq \ell$.

▶ **Lemma 10.** *The storage required by the DAG structure described above is $O(1/\varepsilon^{(d-1)/2})$.*

As mentioned above, by combining Lemmas 4 with 6, it follows that the query time is $O(\log 1/\varepsilon)$ and by Lemma 10 the total space is $O(1/\varepsilon^{(d-1)/2})$, which establish Theorem 8.

While our focus has been on demonstrating the existence of a simple data structure derived from Delone sets, we note that it can be constructed by well-established techniques. While obtaining the best dependencies on ε in the construction time will likely involve fairly sophisticated methods, as seen in the paper of Arya et al. [5], the following shows that there is a straightforward construction.

▶ **Lemma 11.** *Given a convex body $K \subset \mathbb{R}^d$ represented as the intersection of m halfspaces and $\varepsilon > 0$, the above DAG structure for answering ε-APM queries can be computed in time $O(m + 1/\varepsilon^{O(d)})$, where the constant in the exponent does not depend on ε or d.*

Proof. First, we transform K into canonical form, and replace it with an $\frac{\varepsilon}{2}$-approximation K' of itself. This can be done in $O(m + 1/\varepsilon^{O(d)})$, so that K' is bounded by $O(1/\varepsilon^{(d-1)/2})$ halfspaces (see, e.g., [4]). We then build the data structure to solve APM queries to an accuracy of $(\varepsilon/2)$, so that the total error is ε.

Because the number of nodes increases exponentially as we descend to the leaf level, the most computationally intensive aspect of the remainder of the construction is computing the set X_0, a maximal subset of K whose packing ellipsoids $E''_{\delta_0}(x)$ are pairwise disjoint. To discretize the construction of X_0, we observe that by our remarks at the start of Section 2, the Hausdorff distance between K and K_{δ_0} is $\Omega(\delta_0) = \Omega(\varepsilon)$. It follows that each of the ellipsoids $E''_{\delta_0}(x)$ contains a ball of radius $\Omega(\lambda_0\varepsilon) = \Omega(\varepsilon)$. We restrict the points of X_0 to come from the vertices of a square grid whose side length is half this radius. Since K is in canonical form, it suffices to generate $O(1/\varepsilon^{O(d)})$ grid points. By decreasing the value of ε slightly (by a constant factor), it is straightforward to show that any Delone set can be perturbed so that its centers lie on this grid.

Each Macbeath ellipsoid can be computed in time linear in the number of halfspaces bounding K', which is $O(1/\varepsilon^{O(d)})$ [17]. The maximal set is computed by brute force, repeatedly selecting a point x from the grid, computing $E''_{\delta_0}(x)$, and marking the points of the grid that it covers until all points interior to K are covered. The overall running time is dominated by the product of the number of grid points and the $O(1/\varepsilon^{O(d)})$ time to compute each Macbeath ellipsoid. ◀

References

1. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51:606–635, 2004.
2. S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate polytope membership queries. In *Proc. 43rd Annu. ACM Sympos. Theory Comput.*, pages 579–586, 2011.
3. S. Arya, G. D. da Fonseca, and D. M. Mount. Polytope approximation and the Mahler volume. In *Proc. 23rd Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 29–42, 2012.
4. S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate polytope membership queries. *SIAM J. Comput.*, 2017. To appear.
5. S. Arya, G. D. da Fonseca, and D. M. Mount. Near-optimal ε-kernel construction and related problems. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 10:1—10:15, 2017.
6. S. Arya, G. D. da Fonseca, and D. M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete & Computational Geometry*, 58(4):849–870, 2017.
7. S. Arya, G. D. da Fonseca, and D. M. Mount. Optimal approximate polytope membership. In *Proc. 28th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 270–288, 2017.
8. K. Ball. An elementary introduction to modern convex geometry. In S. Levy, editor, *Flavors of Geometry*, pages 1–58. Cambridge University Press, 1997. (MSRI Publications, Vol. 31).
9. I. Bárány. The technique of M-regions and cap-coverings: A survey. *Rend. Circ. Mat. Palermo*, 65:21–38, 2000.
10. I. Bárány and D. G. Larman. Convex bodies, economic cap coverings, random polytopes. *Mathematika*, 35:274–291, 1988.
11. J. L. Bentley, M. G. Faust, and F. P. Preparata. Approximation algorithms for convex hulls. *Commun. ACM*, 25(1):64–68, 1982.
12. A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proc. 23rd Internat. Conf. on Machine Learning*, pages 97–104, 2006.
13. H. Brönnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete Comput. Geom.*, 10:143–155, 1993.

14 C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

15 T. M. Chan. Fixed-dimensional linear programming queries made easy. In *Proc. 12th Annu. Sympos. Comput. Geom.*, pages 284–290, 1996.

16 T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16:369–387, 1996.

17 B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

18 K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. Tenth Annu. Sympos. Comput. Geom.*, pages 160–164, 1994.

19 K. L. Clarkson. Building triangulations using ε-nets. In *Proc. 38th Annu. ACM Sympos. Theory Comput.*, pages 326–335, 2006.

20 R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10(3):227–236, 1974.

21 J. Erickson, L. J. Guibas, J. Stolfi, and L. Zhang. Separation-sensitive collision detection for convex objects. In *Proc. Tenth Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 327–336, 1999.

22 G. Ewald, D. G. Larman, and C. A. Rogers. The directions of the line segments and of the r-dimensional balls on the boundary of a convex body in Euclidean space. *Mathematika*, 17:1–20, 1970.

23 S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

24 S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35:1148–1184, 2006.

25 D. Hilbert. Ueber die gerade linie als kürzeste verbindung zweier punkte. *Mathematische Annalen*, 46:91–96, 1895.

26 R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 798–807, 2004.

27 A. M. Macbeath. A theorem on non-homogeneous lattices. *Ann. of Math.*, 56:269–293, 1952.

28 J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10:215–232, 1993.

29 J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2:169–186, 1992.

30 J. Matoušek. Linear optimization queries. *J. Algorithms*, 14(3):432–448, 1993.

31 F. Nielsen and L. Shao. On Balls in a Hilbert Polygonal Geometry (Multimedia Contribution). In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 67:1–67:4, 2017.

32 A. Papadopoulos and M. Troyanov. *Handbook of Hilbert Geometry*. European Mathematical Society, 2014.

33 E. A. Ramos. Linear programming queries revisited. In *Proc. 16th Annu. Sympos. Comput. Geom.*, pages 176–181, 2000.

34 C. Vernicos and C. Walsh. Flag-approximability of convex bodies and volume growth of Hilbert geometries. HAL Archive (hal-01423693i), 2016. URL: `https://hal.archives-ouvertes.fr/hal-01423693`.

Computing Shortest Paths in the Plane with Removable Obstacles

Pankaj K. Agarwal
Duke University, Durham, NC, USA
pankaj@cs.duke.edu

Neeraj Kumar
University of California, Santa Barbara, CA, USA
neeraj@cs.ucsb.edu

Stavros Sintos
Duke University, Durham, NC, USA
ssintos@cs.duke.edu

Subhash Suri
University of California, Santa Barbara, CA, USA
suri@cs.ucsb.edu

Abstract

We consider the problem of computing a Euclidean shortest path in the presence of *removable* obstacles in the plane. In particular, we have a collection of pairwise-disjoint polygonal obstacles, each of which may be removed at some cost $c_i > 0$. Given a cost budget $C > 0$, and a pair of points s, t, which obstacles should be removed to minimize the path length from s to t in the remaining workspace? We show that this problem is NP-hard even if the obstacles are vertical line segments. Our main result is a fully-polynomial time approximation scheme (FPTAS) for the case of *convex* polygons. Specifically, we compute an $(1+\epsilon)$-approximate shortest path in time $O\left(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon}\right)$ with removal cost at most $(1+\epsilon)C$, where h is the number of obstacles, n is the total number of obstacle vertices, and $\epsilon \in (0,1)$ is a user-specified parameter. Our approximation scheme also solves a shortest path problem for a *stochastic* model of obstacles, where each obstacle's presence is an independent event with a known probability. Finally, we also present a data structure that can answer s–t path queries in polylogarithmic time, for any pair of points s, t in the plane.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Euclidean shortest paths, Removable polygonal obstacles, Stochastic shortest paths, L_1 shortest paths

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.5

Funding Work by Agarwal and Sintos is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation. Work by Suri and Kumar is supported by NSF under grant CCF-1525817.

1 Introduction

We consider a variant of the classical shortest-path problem in the presence of polygonal obstacles, in which the motion planner has the ability to *remove* some of the obstacles to reduce the s–t path length. Formally, let $\mathsf{P} = \{P_1, \ldots, P_h\}$ be a set of h pairwise-disjoint polygonal obstacles in $\mathbb{R}^2$ with n vertices, and let $c_i > 0$ be the cost of removing the obstacle

P_i for $i = 1, \ldots, h$. For a path π in $\mathbb{R}^2$, we define its cost, denoted by $c(\pi)$, to be the sum of the costs of obstacles intersecting π, and its length, denoted by $\|\pi\|$, to be its Euclidean length. Given two points $s, t \in \mathbb{R}^2$ and a budget $C > 0$, we wish to compute a path from s to t of minimum length whose cost is at most C.

This *obstacle-removing* shortest path generalizes the classical obstacle-*avoiding* shortest path problem, by giving the planner an option of essentially "tunneling" through obstacles at some cost. Besides an interesting problem in its own right, it is also a natural formulation of tradeoffs in some motion planning settings. For instance, it might be beneficial to remove a few critical blockages in a workspace to significantly shorten an often traveled path, just as an urban commuter may strategically pay money to use certain toll roads or bridges to avoid traffic obstacles. In general, our model with removable obstacles is useful for applications where one can adapt the environment to enable better paths such as urban planning or robot motion planning in a warehouse setting. The problem also generalizes the recent work on obstacle-*violating* paths [25, 18], in which the planner is allowed to enter the forbidden space (obstacles) a fixed *number* of time. For instance, in [25], a shortest s–t path inside a simple polygon is desired, but the path is allowed to travel outside the polygon once. In [18], a shortest path among disjoint convex polygonal obstacles is desired, but is allowed to travel through at most k obstacles. The latter problem is also an obstacle-removing shortest path where *at most k obstacles* can be removed, namely, each obstacle removal has cost 1 and planner's budget is k. We will call this the *cardinality* version of the obstacle-removal to distinguish it from our cost-based model of obstacle removal.

The obstacle removal problem also has a natural connection to path planning under uncertainty. Imagine, for instance, a workspace with n obstacles, the presence of each obstacle is a random event. That is, the presence of the ith obstacle is determined by a Bernoulli trial with (independent) probability β_i. A natural approach to planning a s–t path in such a workspace is to search for a path that is both short and obstacle-free with high probability. Given a desired probability of success β, we can ask: what is the shortest path from s to t that is obstacle free with probability at least β. This problem is easily transformed into our obstacle removal problem where the obstacle probabilities are mapped to obstacle removal cost, and β is mapped to the cost budget C.

Our results. We first show that the obstacle-removing shortest path problem is NP-hard for polygonal obstacles in the plane, even if obstacles are vertical line segments by reducing the well-known Partition problem to it. This is in contrast with the cardinality version of the problem, which can be solved exactly in $O(k^2 n \log n)$ time [18].

Our main result is a fully-polynomial time approximation scheme (FPTAS) when each obstacle is a *convex* polygon. We first define the notion of the *viability* graph G, which is an extension of the well-known visibility graph [11, 13], for geometric paths that can cross obstacles. Using the viability graph, we present a simple algorithm that returns a path with length at most the optimal[1] but cost at most $(1+\epsilon)C$. The approximation algorithm, while simple, has a worst-case time complexity $\Theta(\frac{n^3}{\epsilon}\,\mathrm{polylog}(n))$. Then, we develop a framework for a more efficient and practical approximation algorithm, which also results in a number of related results. Specifically, for any constant $\epsilon > 0$, we can compute a $(1+\epsilon)$-approximate shortest path whose total removal cost is at most $(1+\epsilon)C$ in time $O\left(\frac{nh}{\epsilon^2}\log n \log \frac{n}{\epsilon}\right)$, where h is the number of obstacles and n is the total number of vertices in the obstacles. The main idea behind the improvement is to construct a *sparse* viability graph, with only $O(\frac{n}{\epsilon}\log n)$

[1] The optimal length is always with respect to the budget C.

edges. This approximation scheme immediately gives a corresponding result for the uncertain model of obstacles (see Section 5).

The approximation scheme, as a byproduct, also solves the exact L_1 norm shortest path problem in the *cardinality* model of obstacle removal: that is, in $O(kn \log^2 n)$ time, we can decide which k obstacles to remove for the shortest s–t path, which is roughly a factor of k faster than the L_2-norm result of [18]. Alternatively, we can also decide which k obstacles to remove so that the shortest s–t path has length at most $(1+\epsilon)$ times optimal in $O(\frac{kn}{\epsilon} \log^2 n)$ time. This is again faster than the result from [18] for constant ϵ, if $k = \Omega(\log n)$.

We also construct query data structures for answering approximate obstacle removal shortest path queries. If the source s is fixed (one point queries), we construct a data structure of size $O(\frac{nh}{\epsilon^2} \log n)$ such that, given a query point t, it returns a s–t path of length $(1+\epsilon)$ times the optimal with cost at most $(1+\epsilon)C$ in time $O(\frac{1}{\epsilon} \log^2 n + k_{st})$, where k_{st} is the number of edges in the path. The data structure size can be improved to $O(\frac{n}{\epsilon^2} \log n \log \frac{h}{\epsilon})$ if we only return the length of the path. If both points s, t are given in the query (two point queries), the data structure has size $O(\frac{n^2 h}{\epsilon^3} \log^2 n)$, and the query time is $O(\frac{1}{\epsilon^2} \log^2 n + k_{st})$. The size of the data structure can also be improved to $O(\frac{n^2}{\epsilon^3} \log^2 n \log \frac{h}{\epsilon})$ if we only return the length of the path.

Related work. The problem of computing a shortest path in the presence of polygonal obstacles in the plane is a very well studied problem in computational geometry. See the books [16, 31], survey paper [28], recent papers [9, 6, 10, 8, 20], and references therein for a sample of results. In the classical shortest path problem, obstacles are impenetrable, that is, the shortest path must avoid all the obstacles. Our problem considers a more general scenario where the obstacles can be removed by paying some cost and falls in the broad category of geometric optimization problems where some constraints can be violated [2, 30, 26, 17].

Our problem is also closely related to the problem of computing a shortest path through weighted polygonal regions [23, 24, 27] where the length of a path is defined as the weighted sum of Euclidean or L_1 lengths of the subpaths within each region. However, in our setting there is only a one-time fixed cost for passing through a region, and therefore does not depend on the length of the subpath that lies inside the region.

The stochastic formulation of our problem is also related to some shortest path problems under uncertainty [14, 15, 22, 29]. However, these results assume existence of a graph whose edges have either an existence probability or a distribution over their lengths. In contrast, our definition is purely geometric where the existence of obstacles is an uncertain event. Our problem can also be seen as a variant of geometric bi-criteria shortest path problem [1, 5, 33, 34, 35], as our objective is to compute the shortest path with a constraint on the total cost of obstacles that we remove.

Finally, for most geometric shortest path problems, there are efficient data structures to answer shortest path queries. For instance, the shortest path map [19] has linear size and can answer Euclidean shortest path queries with a fixed source in $O(\log n)$ time. If both s, t are part of the query, quadratic space data structures [4, 21] exist for L_1 shortest path queries and super quadratic space data structures [12] for L_2 shortest path queries. Similar results exist for rectilinear shortest path queries among disjoint weighted rectilinear and convex obstacles [4, 7], and for bi-criteria shortest path problems [5, 33, 35].

Overall, our algorithms entail new techniques because (i) in our problems, paths are allowed to pass through obstacles, (ii) the cost function in our bi-criteria optimization can be quite general and not necessarily a metric.

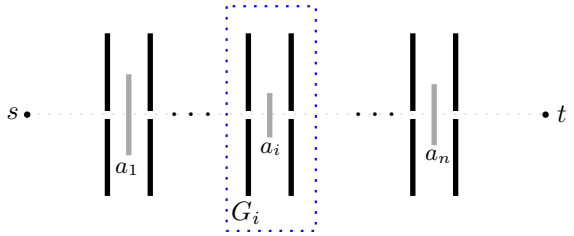


Figure 1 Reduction from Partition. The gray segment in the obstacle group G_i has length a_i and can be crossed by paying a cost a_i. The tall segments are drawn in black and are placed $\pm\delta$ apart from their corresponding gray segment.

2 NP-hardness

Consider the decision version of the obstacle-removing shortest-path problem: Given a set P of pairwise-disjoint obstacles along with the cost of each object being removed, two points $s, t \in \mathbb{R}^2$, and two parameters $C, L > 0$, is there a path from s to t of length at most L and cost at most C?

We prove the hardness by a simple reduction from the well-known NP-complete problem Partition. An instance of Partition is a set of n positive integers $A = \{a_1, a_2, \ldots, a_n\}$, and the problem is to decide whether A can be partitioned into two sets A_1 and A_2 such that $W(A_1) = W(A_2) = \frac{1}{2}W(A)$, where $W(S)$ is the sum of the integers in S. We place the source s at $(0, 0)$ and destination t at $(n + 1, 0)$ on the x-axis. We also set $C = \frac{1}{2}W(A)$, $L = \frac{1}{2}W(A) + (n + 1)$ and define a parameter $\delta = \frac{1}{8n}$. For each $i \leq n$ we create a group of obstacles, denoted G_i, which consists of five vertical line segments placed close to each other in the following way. (See also Figure 1.)

- The middle segment e_i^m has length a_i, and has its midpoint on the x-axis. The coordinates of its endpoints are $(i, -a_i/2), (i, a_i/2)$. The cost of this obstacle is a_i.
- At x-coordinates $i - \delta$ and $i + \delta$ we place two vertical segments e_i^l and e_i^r symmetrically along the x-axis – each with point-sized holes on the x-axis and length $2(L + 1)$. The point sized holes split the segment e_i^l (resp. e_i^r) into two disjoint *tall* segments e_i^{lu}, e_i^{ld} (resp. e_i^{ru}, e_i^{rd}), of length $(L + 1)$. Each of these segments has cost $(C + 1)$.

▶ **Lemma 1.** *The set A can be partitioned into two equal-weight subsets if and only if there is a path from s to t of length at most L and cost at most C.*

We thus obtain the following:

▶ **Theorem 2.** *Let P be a set of n disjoint polygonal obstacles in a plane, where each obstacle $P_i \in \mathsf{P}$ has an associated removal cost c_i. Given a source and destination pair of points s, t, a removal budget C and a length L, the problem of deciding if there is a s–t path with cost at most C and length at most L is* NP*-hard.*

3 A Simple $(1 + \epsilon)$-Approximation Algorithm

In this section, we propose a simple polynomial-time approximation scheme for the problem. We begin by noting that an obstacle-removing shortest path only turns at obstacle vertices and crosses the boundary of an obstacle at most twice. While these properties follow easily due to the convexity of P and basic geometry, they are crucial for our algorithms.

The algorithm constructs a *viability graph* $G = (V, E)$, whose nodes are all the obstacle vertices along with s and t. Thus, $|V| = n + 2$. The edges of E correspond to pair of nodes (u, v) for which the line segment uv passes through obstacles of total cost at most C, the cost budget. For each edge $e \in E$, we associate two parameters: cost $c(u, v)$ and length $\|uv\|$, where $c(u, v)$ is the cost of the segment uv. In the worst-case G has $\Theta(n^2)$ edges. It is important to note that the cost of a path π_{st} in a viability graph is defined as the sum of the costs of its edges, whereas the cost of π_{st} in the plane is defined as sum of costs of all obstacles that it goes through. Moreover, the cost of a path in the plane is at most its cost in the viability graph. If the path crosses each obstacle at most once (which is the case for shortest path among convex obstacles), these two costs are the same.

The following algorithm shows how to compute an approximately optimal path in this viability graph. The main idea is that we construct copies of the vertices and the edges of G to convert the multi-objective problem to a single-objective problem.

Let $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$. To simplify the approximation error analysis, we first scale all the costs by κ/C, so that the new target cost is κ. We now construct an auxiliary graph $G' = (V', E')$, with $O\left(\left\lceil \frac{2\kappa|V|}{\epsilon} \right\rceil\right)$ nodes and $O\left(\left\lceil \frac{2\kappa|E|}{\epsilon} \right\rceil\right)$ edges, *whose edges only have the length parameter but not the cost parameter*, as follows. We create $\left\lceil \frac{2\kappa}{\epsilon} \right\rceil + 1$ copies $v_0, v_{\frac{\epsilon}{2}}, v_\epsilon, v_{\frac{3}{2}\epsilon} \ldots, v_\kappa$, for each $v \in V$. Then, for each edge $(u, v) \in E$ with cost c and for each $0 \le i \le \lceil 2\kappa/\epsilon \rceil$, we add the edge $(u_{i\frac{\epsilon}{2}}, v_{j\frac{\epsilon}{2}})$, where $j \le \lceil 2\kappa/\epsilon \rceil$ is the maximum integer with $j\frac{\epsilon}{2} \le i\frac{\epsilon}{2} + c$. All these edge copies have the same length as edge (u, v)—the cost parameter is now implicitly encoded in the edge copies. Finally we add two new vertices s and t in G' and connect them to all s_i and t_i respectively with zero length edges, for $0 \le i \le \lceil 2\kappa/\epsilon \rceil$. We now find the *minimum length* path π from s to t in G', say, using Dijkstra's algorithm, and argue that π is our approximation path.

▶ **Theorem 3.** *Let* P *be a set of* h *convex obstacles with* n *vertices,* s, t *be two obstacle vertices, and* $C \in \mathbb{R}$ *be a parameter. Let* L^* *also be the length of the shortest* s*–*t *path with cost at most* C*, and let* $G = (V, E)$ *be a viability graph induced by this workspace. If there exists a path* π^* *of length at most* αL^* *with* $\alpha \ge 1$ *and cost at most* C *in the graph* G*, then a* s*–*t *path* π *with length at most* αL^* *and cost at most* $(1 + \epsilon)C$ *can be computed in time* $O\left(\frac{\kappa}{\epsilon}(|E| + |V| \log \frac{|V|}{\epsilon})\right)$*, where* $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$ *and* $0 < \epsilon < 1$ *is a parameter.*

Proof. First, we construct the auxiliary graph G' as described above. Next, we construct a path π' in G' corresponding to the path π^* in G by mapping edges of π^* to edges in G'. More precisely, let $e = (s, v)$ be the first edge in π^* and let c_e be its cost. Now let $c = 0$ and c' be the value obtained by *rounding down* c_e to the nearest multiple of $\frac{\epsilon}{2}$. We map e to the edge $(s_c, v_{c'})$ in G'. Setting $c = c'$, we repeat the process for all edges in π^*. This gives us the path π' in G' that has the length same as that of π^* (at most αL^*). Clearly, the s–t path π computed using Dijkstra's algorithm on G' must also have length at most αL^*. Moreover, since (scaled) rounded cost of any s–t path in G' is at most κ, the rounded cost of π is also at most κ. Now we only need to bound its original (pre-rounded) cost.

Let C_R be the true (pre-rounded) cost of the path π in the plane and C_A its rounded cost in G'. The approximation error in the cost (due to rounding) is at most $\epsilon/2$ for each obstacle that π passes through, and so if $\bar{k}$ is the number of obstacles π crosses, we have the upper bound $C_R \le C_A + \bar{k}\epsilon/2$. Since $C_A \le \kappa$, we have $C_R \le \kappa + \bar{k}\epsilon/2$. We can bound $\bar{k}$ by considering the following two cases. If $\kappa = C/\min_i c_i$, the minimum cost of an obstacle is 1, and so for each obstacle crossed, the path π incurs a cost of least $1 - \epsilon/2$. Therefore, $\bar{k} \le \frac{\kappa}{1-\epsilon/2}$ and $C_R \le \kappa + \frac{\kappa}{1-\epsilon/2} \cdot \epsilon/2 \le \frac{1}{1-\epsilon/2}\kappa \le (1 + \epsilon)\kappa$. Otherwise, we have $\kappa = h$, which trivially implies $\bar{k} \le \kappa$ since h is the total number of obstacles.

In conclusion, we have $C_R \leq (1+\epsilon)\kappa$, whose pre-scaled value is $\frac{(1+\epsilon)\kappa}{(\kappa/C)} = (1+\epsilon)C$, as claimed. Finally, the time complexity is dominated by an invocation of Dijkstra's algorithm on the graph G', which has $O(|V|\kappa/\epsilon)$ nodes and $O(|E|\kappa/\epsilon)$ edges. ◀

If G is the viability graph constructed in this section then it always contains the shortest s–t path with cost at most C, i.e. $\alpha = 1$. Hence, by applying Theorem 3 to G we get a path of at most the optimum length and cost at most $(1+\epsilon)C$ in $\Omega(\frac{n^3}{\epsilon})$ time.

In the next section, we show that if we also allow an $(1+\epsilon)$ approximation of the path length, we can improve the running time by roughly an order of magnitude.

4 A Faster $(1+\epsilon)$-Approximation Algorithm

In this section, we describe our algorithm for sparsifying the graph $G = (V, E)$. We augment the graph by adding some vertices so that the number of viability edges can be sharply reduced, while approximately preserving the path lengths within the cost budget. Throughout the following discussion, we will respect the cost budget C, and only allow the path lengths to increase slightly. With that in mind, we use the notation $d_G(u, v)$ to denote the length of the shortest path in G from u to v whose cost is at most C. In this section we only use the definition of the cost of a path with respect to a viability graph. Recall that the cost of a path in a graph is the sum of the costs of the edges in the path.

Our sparse graph $H_\epsilon = (X_\epsilon, T_\epsilon)$ is defined for any $\epsilon > 0$, with $V \subseteq X_\epsilon$, and satisfies the following two conditions:

1. $d_G(u, v) \leq d_{H_\epsilon}(u, v) \leq (1+\epsilon)d_G(u, v)$ for all pairs $u, v \in V$.
2. The number of vertices and edges is $O(\frac{n}{\epsilon}\log n)$, that is, $|X_\epsilon|, |T_\epsilon| = O(\frac{n}{\epsilon}\log n)$.

We construct H_ϵ in two stages. In the first stage we construct a graph $H = (X, \Gamma)$ with $X \supseteq V$, $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2}d_G(u, v)$ for all $u, v \in V$. Next, we make $O(1/\epsilon)$ "copies" of H and combine them to construct H_ϵ. Once the graphs H and H_ϵ are constructed, we use the machinery of the previous section, namely Theorem 3, to efficiently find the approximately optimal shortest path within the cost budget.

Recall that all the obstacles in our input are convex, and therefore the shortest path in G does not cross the boundary of an obstacle more than twice. To avoid degenerate cases, we assume that all obstacle vertices are in general position, namely, no three vertices are collinear and all obstacles have non-zero area. We can, therefore, simplify the problem by replacing all the obstacles by their constituent boundary segments, where each obstacle vertex is assigned to its incident segment in the clockwise order. We now allocate the "obstacle removal" cost to these segments as follows: if c_i is the removal cost of obstacle i, then we allocate cost $c_i/2$ to each boundary segment of obstacle i. This ensures that any shortest path crossing the ith obstacle incurs a cost of c_i, while allowing us to reason about the geometry of just line segment obstacles.

We describe the construction of the sparse viability graph by explaining how to sparsify the "neighborhood" of an obstacle vertex, say, p. That is, we show which additional vertices are added and which viability edges are incident to p in the final sparse graph H. To simplify the discussion, we assume that p is at the origin, and we only discuss the edges incident to p that lie in the positive (north-east) quadrant; the remaining three quadrants are processed in the same way.

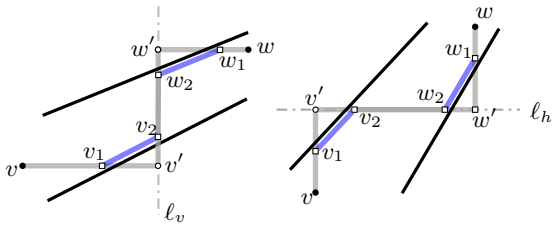


Figure 2 Steiner vertices due to vertical (left) and horizontal (right) split lines. Projections are shown with white dots, bypass vertices as squares, bypass edges shown in blue have cost zero.

4.1 An $O(1)$-Approximation Algorithm

In this subsection we describe the construction of $H = (X, \Gamma)$ such that $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$.

For a segment pq we use $\|pq\|_1$ to denote its L_1-length, i.e., $\|pq\|_1 = |x_p - x_q| + |y_p - y_q|$, where $p = (x_p, y_p)$ and $q = (x_q, y_q)$. For a polygonal path $\pi = p_0 p_1 \dots p_k$, we use $\|\pi\|_1$ to denote its L_1-length, i.e., $\|\pi\|_1 = \sum_{i=1}^{k} \|p_{i-1} p_i\|_1$. We note that $\|\pi\|_1 \leq \sqrt{2}\|\pi\|$. We will construct a graph $H = (X, \Gamma)$ with the following property: For a pair of vertices $u, v \in V$ if G contains a path π from u to v of cost at most C, H contains a path $\bar{\pi}$ from u to v of cost at most C such that $\|\bar{\pi}\|_1 \leq \|\pi\|_1$. Hence $\|\bar{\pi}\| \leq \sqrt{2}\|\pi\|$ and thus $d_H(u, v) \leq \sqrt{2} d_G(u, v)$.

We are now ready to describe the algorithm for constructing H. It is a simple recursive algorithm and consists of the following steps:

1. Let x_m be the median x-coordinate of the points in V. We consider the vertical *split line* $\ell_v : x = x_m$ that partitions V into two almost equal-sized subsets V_l and V_r.
   - **a.** For each point $v \in V$, consider its projection $v' = (x_m, v_y)$ on the line ℓ_v. If $c(v, v') \leq C$, then add the *projection* vertex v' to X and the corresponding edge $e = (v, v')$ to Γ with length $\|vv'\|_1$ and cost $c(v, v')$.
   - **b.** Let s' be the first obstacle segment with positive slope that the projection segment vv' intersects. If s' intersects the split line ℓ_v, we add *bypass* vertices and edges to H as follows. Let v_1 be the point where vv' intersects s', and let v_2 be the point where s' intersects ℓ_v. We add bypass vertices v_1, v_2 on the segment s'. If v_2 lies above v_1, the bypass vertices are considered to be above the segment s', otherwise they are considered below the segment s'. (See also Figure 2.) We add the edges (v, v_1) and (v_1, v_2) to Γ with lengths $\|vv_1\|_1, \|v_1 v_2\|_1$ and costs $c(v, v_1), c(v_1, v_2)$, respectively. Observe that $c(v_1, v_2) = 0$.
   - **c.** We repeat the procedure above for the first negative slope segment that vv' intersects.
   - **d.** For two consecutive Steiner vertices w, w' (projection or bypass) on ℓ_v, if $c(w, w') \leq C$, then add the edge $e = (w, w')$ to Γ with length $\|ww'\|_1$ and cost $c(w, w')$.
   - **e.** Recurse on the subsets V_l and V_r until $|V_l|, |V_r| \leq 1$.
2. Repeat the above process but this time using median y-coordinate y_m and a horizontal split line ℓ_h at $y = y_m$.
3. We add edges between consecutive vertices on the boundary of obstacles with cost 0.

At each recursive step of our algorithm, we need to find the first positive (negative) slope obstacle segment intersected by the projection segment vv', and compute the cost of all edges we add. In order to find the first positive (negative) slope segment say s', we can simply perform a point location query in $O(\log n)$ time [32] on positive (negative) slope segments. If s', intersects both the projection segment vv' and the split line passing through v', we add the bypass vertices. For computing the edges costs, observe that bypass edges and the edges

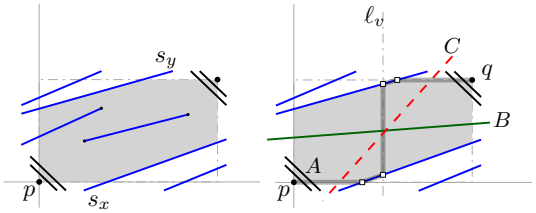


Figure 3 The region R_{pq} is shown shaded. If R_{pq} does not contain obstacle vertices, the type A, B, C obstacle segments that may intersect R_{pq} are shown on the right. Observe that type B and type C segments cannot both exist in R_{pq}.

on the boundary of obstacles have both cost zero, and all other edges are either horizontal or vertical line segments, so we just need to compute the total cost of obstacle segments intersected by an axis aligned segment. We show how to do this for a horizontal projection segment vv' and all other cases follow similarly. We preprocess all the obstacle segments in a segment tree based data structure S. Using fractional cascading and increasing the fan-out of the segment tree [3, 32], a (weighted) counting query runs in $O(\log n)$ time. During each recursive call, we simply query S to compute the cost of the segment vv'. However, we need to be careful in including the cost of the obstacle segment that v lies on. More precisely, if P_i is the obstacle incident to v, we include the cost $c_i/2$ to the cost of segment vv' only if vv' intersects the interior of P_i (which we can decide in constant time).

We can easily obtain the following lemma.

▶ **Lemma 4.** *Every input vertex adds Steiner vertices on $O(\log n)$ split lines. Moreover, graph H has size $O(n \log n)$ and can be constructed in $O(n \log^2 n)$ time.*

It is important to note here that a similar recursive algorithm was first used by Clarkson et al. [13] to compute L_1 shortest obstacle-avoiding paths in the plane – each vertex was projected on $O(\log n)$ split lines and on the obstacle closest to it in all four directions. This was enough to capture obstacle-avoiding shortest paths (as they lie entirely in free space) but since obstacle-removing shortest paths can also go through obstacles, things get quite complicated. In particular, it is not clear that which of the $O(n)$ nearby obstacles (in each direction) should a vertex be projected on. We address this challenge in Step 1b of our algorithm by adding bypass vertices. Since we need to guarantee that the sparsification preserves the L_1 length as well as the cost of the shortest path, our correctness argument is quite different and can be viewed as a more general form of the result by [13].

4.2 Proof of Correctness

We now prove that $d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$. More precisely, if we set the length of each edge $e = (u, v)$ in G to be $\|uv\|_1$, then we show that $d_H(u, v) \leq d_G(u, v)$. We basically show that for any edge $e = (u, v)$ in G there is a path π_e from u to v in H such that $c(\pi_e) \leq c(u, v)$ and $\|\pi_e\|_1 \leq \|uv\|_1$. This claim is established in Lemma 7, whose proof relies on the following Lemmas 5 and 6.

For convenience, we introduce the notion of the region R_{pq} defined by two obstacle vertices $p, q \in V$. Let $\bar{R}_{pq}$ be the rectangle with p and q as lower left and upper right corners respectively. Now, let s_x (resp. s_y) be the first obstacle segment of positive slope that intersects the two sides of $\bar{R}_{pq}$ below (resp. above) the diagonal pq. We define $R_{pq} = \bar{R}_{pq} \setminus (\mathcal{B}(s_x) \cup \mathcal{A}(s_y))$, where $\mathcal{B}(s_x)$ is the area below segment s_x, and $\mathcal{A}(s_y)$ is the area above s_y. If a segment s_x or s_y does not exist then $\mathcal{B}(s_x) = \emptyset$ and $\mathcal{A}(s_y) = \emptyset$. (See also Figure 3.)

▶ **Lemma 5.** *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} does not contain an obstacle vertex, then there exists a path π_{pq} in H that is entirely contained in R_{pq} such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(p, q)$.*

Proof. Since R_{pq} does not contain any obstacle vertex there are only three types of obstacle segments that intersect R_{pq}. (See also Figure 3.)

1. *Type A* : these obstacle segments have negative slope and intersect both vertical and horizontal segments of R_{pq} adjacent to either p or q.
2. *Type B* : obstacle segments that intersect both vertical segments of R_{pq}.
3. *Type C* : obstacle segments that intersect both horizontal segments of R_{pq}.

It is easy to see that segments of type B and C cannot both exist in R_{pq} because the obstacle segments are non-intersecting. From the construction of H there is always a vertical and a horizontal split line between two obstacle vertices. Let ℓ_v (ℓ_h) be the first vertical (horizontal) split line in the recursion that we consider between the vertices p, q. There are three cases.

- *Only Type A segments exist in R_{pq}.* This case is taken care by the Steiner vertices on the vertical (or horizontal) split line ℓ_v. More precisely, ℓ_v may intersect both s_x and s_y, one of them, or even neither of them. We show what happens in the case where ℓ_v intersects both s_x and s_y and the other cases follow easily. Since there are no obstacle vertices in R_{pq}, s_x, s_y are the first positive slope segments intersected by the projections of p, q on ℓ_v. So we have created bypass vertices p_1, p_2 and q_1, q_2 on s_x, s_y. The path π_{pq} is defined as $\pi_{pq} = pp_1p_2q_2q_1q$ and it is easy to see that $\|\pi_{pq}\|_1 = \|pq\|_1$. Moreover, both π_{pq} and the edge pq cross one time the same set of obstacle segments (only type A), so we have that $c(\pi_{pq}) = c(p, q)$.
- *Type B segments exist in R_{pq}.* In this case, observe that type B edges do not intersect with the horizontal projection segments adjacent to p and q on the vertical split line, and therefore we can use the exact same path π_{pq} as the previous case. The cost of the type B segments needs to be included but since the edge pq must cross these segments, we have that $c(\pi_{pq}) = c(p, q)$.
- *Type C segments exist in R_{pq}.* This case is symmetric to the previous case using the *horizontal* split line ℓ_h. ◀

▶ **Lemma 6.** *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} contains one or more obstacle vertices, then there exists an obstacle vertex $r \in R_{pq}$ such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$, and $c(p, r) + c(r, q) \leq c(p, q)$.*

Proof. We prove the lemma by exhibiting a vertex r such that (i) the triangle Δprq does not contain any other obstacle vertex, and (ii) no obstacles segment intersects the interiors of both pr and rq. Such a choice of r suffices for our proof since $r \in R_{pq}$ implies that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and we get $c(p, r) + c(r, q) \leq c(p, q)$ because any obstacle segment crossing either pr or rq must also cross pq, otherwise that obstacle segment would terminate inside the triangle which contradicts the choice of r. Next, we show how to find such a vertex. We restrict our search for this vertex r in a convex polygon $T_{pq} \subseteq R_{pq}$ which we construct in the following way. (See also Figure 4.) Observe that the diagonal pq divides the region R_{pq} into two subsets – one above and one below it. We consider the subset R'_{pq} that contains at least one obstacle vertex. Since, R_{pq} contains at least one obstacle vertex, such a subset always exists. Without loss of generality, we can assume that R'_{pq} lies above pq. Now, let S_{pq} be the set of all obstacle segments that intersect a vertical or a horizontal segment of the boundary $\partial R'_{pq}$, and let $s_p, s_q \in S_{pq}$ be the segments that intersect $\partial R'_{pq}$ closest to p and q respectively. From the endpoints of s_p, s_q that lie in R'_{pq}, let w be the endpoint closest to the segment pq. Moreover, let l_w be the line parallel to pq that passes through w. Now we simply

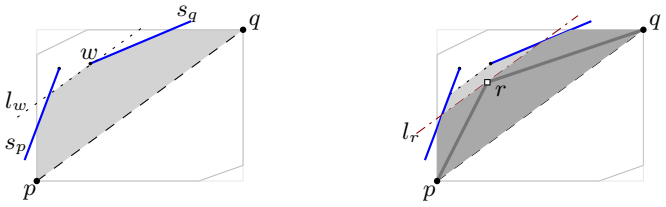


Figure 4 The region T_{pq} is shown shaded on left. If $r \in T_{pq}$ is the vertex closest to pq, then the region $T'_{pq} \subseteq T_{pq}$(shown shaded in dark on right) cannot contain an obstacle vertex.

clip off the region of R'_{pq} that lies above l_w. More precisely, this gives us the quadrilateral $R''_{pq} = R'_{pq} \setminus \mathcal{A}(l_w)$, where we use $\mathcal{A}(s)$ for the region above segment s. Finally, we define the convex polygon $T_{pq} = R''_{pq} \setminus (\mathcal{A}(s'_p) \cup \mathcal{A}(s'_q))$, where s'_p, s'_q are the subsegements of s_p, s_q respectively that lie inside the quadrilateral R''_{pq}.

From the set of obstacle vertices that lie *inside or on the boundary* of T_{pq}, we choose the vertex r to be the one that minimizes the area of the triangle Δprq, or equivalently, be the one that has the minimum distance from the segment pq. Observe that the boundary of region T_{pq} contains the obstacle vertex w, so we will always find one such r. It is easy to see that the triangle Δprq is a subset of T_{pq} and does not contain an obstacle vertex or else it would not have the minimum area. It remains to show that there cannot be an obstacle segment that crosses both pr and rq. To this end, let l_r be a line parallel to pq passing through r. Observe that the region $T'_{pq} = T_{pq} \setminus \mathcal{A}(l_r)$, i.e., the region in T_{pq} that lies below l_r, cannot contain an obstacle vertex by the choice of r. So any obstacle segment s_j that crosses both pr and rq must intersect $\partial R'_{pq}$ at either the vertical segment between p and s_p or the horizontal segment between s_q and q which is a contradiction. (See also Figure 4.) ◀

Finally, we prove the main result of this section.

▶ **Lemma 7.** *Let (p, q) be an edge in G with cost $c(p, q)$. There is a path $\pi_{pq} \in H$ such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) \leq c(p, q)$. Moreover, the path π_{pq} lies in the region R_{pq}.*

Proof. We prove this by induction on the number of obstacle vertices in the region R_{pq}. Our base case is when the region R_{pq} does not contain an obstacle vertex. Applying Lemma 5 gives us the desired path π_{pq} in H. For the inductive step, let j be the number of obstacle vertices in the region R_{pq} and assume that the lemma holds for all edges (u, v) such that the region R_{uv} contains $i < j$ obstacle vertices. Using Lemma 6 we find an intermediate vertex r such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(p, r) + c(r, q) \leq c(p, q)$. This gives us two disjoint subregions $R_{pr} \subset R_{pq}$ and $R_{rq} \subset R_{pq}$ each with at least one less obstacle vertex than the region R_{pq}. By our induction hypothesis, we get the disjoint subpaths π_{pr} from p to r and π_{rq} from r to q in H. We then join these two paths at vertex r to obtain path π_{pq} that lies within the region R_{pq}. Moreover, we have that $\|\pi_{pq}\|_1 = \|\pi_{pr}\|_1 + \|\pi_{rq}\|_1 = \|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(\pi_{pr}) + c(\pi_{rq}) \leq c(p, r) + c(r, q) \leq c(p, q)$. ◀

4.3 An $(1 + \epsilon)$-Approximation Algorithm

We now describe how to use the preceding construction to define our final sparse graph H_ϵ. A direction in $\mathbb{R}^2$ can be represented as a unit vector $\mathbf{u} \in \mathbb{S}^1$. Let $\mathsf{N} \subset \mathbb{S}^1$ be a set of $O(1/\epsilon)$ unit vectors such that the angle between two consecutive points of N is at most ϵ. For each $\mathbf{u} \in \mathsf{N}$, we construct a graph $H^{\mathbf{u}}$ by running the algorithm in Section 4.1 but regarding $\mathbf{u}$ to be the y axis — i.e., by rotating the plane so that $\mathbf{u}$ becomes parallel to the y-axis and

measure L_1-distance in the rotated plane. Set $H_\epsilon = \bigcup_{\mathbf{u}\in\mathsf{N}} H^{\mathbf{u}}$. Notice that the number of vertices and edges in H_ϵ is $O(\frac{n}{\epsilon}\log n)$. The following lemma follows easily by the discussion above.

▶ **Lemma 8.** *For any pair $u, v \in V$, we have that $d_{H_\epsilon}(u,v) \le (1+\epsilon)d_G(u,v)$.*

From the above lemma, it follows that the graph H_ϵ preserves pairwise shortest path distances within a factor of $(1+\epsilon)$ and at most the same cost with graph G. Let L^* be the length of the shortest s–t path in the plane that has cost at most C. Since there exists a s–t path of length at most L^* and cost at most C in the viability graph G, there exists a s–t path in H_ϵ of length $(1+\epsilon)L^*$ and the same cost. Applying Theorem 3 with $\alpha = (1+\epsilon)$ on H_ϵ gives the following result.

▶ **Theorem 9.** *Let P be a set of h convex polygonal obstacles with n vertices, s, t be two obstacle vertices and $C \in \mathbb{R}$ be a parameter. If L^* is the length of the shortest s–t path with cost at most C, a s–t path with length at most $(1+\epsilon)L^*$ and cost at most $(1+\epsilon)C$ can be computed in $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ time.*

5 Shortest Path Queries

We now describe a near-linear space data structure to answer approximate distance queries from a fixed obstacle vertex s subject to the obstacle removal budget in $O(\frac{1}{\epsilon}\log^2 n)$ time. The data structure is then extended to handle two-point shortest path queries in $O(\frac{1}{\epsilon^2}\log^2 n)$ time with near-quadratic space.

The key idea relies on the following observation. Without loss of generality, assume that the points s and t lie in the exterior of all obstacles and let us also assume that s, t were part of the input. Now consider the shortest s–t path in the graph H_ϵ and let t' be the vertex preceding t in this path. It is easy to see that t' must be a Steiner vertex (projection or bypass) as there are no direct edges in H_ϵ between two input vertices that do not lie on the same obstacle. All such edges must cross some split line at Steiner vertices. Therefore, the last edge (t', t) in the path is the segment obtained by projecting t on some split line ℓ. Now, suppose we have precomputed the paths to all Steiner vertices on all split lines, then we can find the shortest path to t by simply finding the neighbor of t' on ℓ. Using Lemma 4, we know that t can be projected on $O(\frac{1}{\epsilon}\log n)$ split lines, which gives $O(\frac{1}{\epsilon}\log n)$ choices for ℓ.

Preprocessing. We apply the algorithm preceding Theorem 3 on the graph H_ϵ that we constructed in the previous section. More precisely, first we multiply the cost of all obstacles by h/C so that the target cost becomes h. Next we create an auxiliary graph H'_ϵ with $O(\frac{h}{\epsilon})$ copies of each vertex in H_ϵ. Running Dijkstra's algorithm on H'_ϵ with source s computes a shortest path to each vertex in H'_ϵ. Now for each vertex v in H_ϵ, we maintain arrays $dist_v, pred_v$ each with size $1 + \frac{h}{\epsilon} = O(\frac{h}{\epsilon})$. We store the length of the shortest path found by Dijkstra's algorithm from s to $v_{i\epsilon}$ (i-th copy of vertex v) at $dist_v(i)$ and its predecessor in $pred_v(i)$. In addition, for each direction $\mathbf{u} \in \mathsf{N}$ that we defined in the previous section we maintain two data structures:

- A segment tree [3] based data structure $S_{\mathbf{u}}$ that we also used in Section 4.1 to compute the cost of an axis aligned segment in $O(\log n)$ time.
- A balanced search tree $T_{\mathbf{u}}$ over all the vertical (resp. horizontal) split lines, which is basically the recursion tree corresponding to the algorithm from Section 4.1. More precisely, the root of $T_{\mathbf{u}}$ is the split line ℓ_m (at the median x-coordinate x_m), and the left and right children are the split lines added during recursive processing of points to the left and right of ℓ_m respectively.

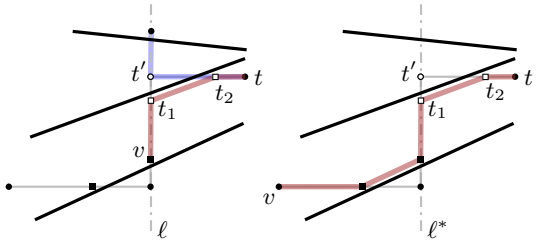


Figure 5 Computing path from a query point t to one of the vertices in H_ϵ – using a split line that already exists in H_ϵ (left) and using a new split line ℓ^* added at query time (right). The suffix path π_{vt} is shown shaded in red.

Moreover, for every split line ℓ, we maintain a search tree over all the Steiner vertices that lie on ℓ. Overall, our data structure consists of all arrays $dist_v, pred_v$, $O(\frac{n}{\epsilon})$ search trees, and $O(\frac{1}{\epsilon})$ segment trees $S_{\mathbf{u}}$. The size of the data structure is $O(\frac{nh}{\epsilon^2}\log n)$ and the preprocessing time is $O(\frac{nh}{\epsilon^2}\log n \log \frac{n}{\epsilon})$.

Query. The query procedure consists of two parts. Given the target query point t, we first find a subset of $O(\frac{1}{\epsilon}\log n)$ split lines L that we need to search. Next, for each line $\ell \in L$, we find the Steiner vertex t' created by projecting t on ℓ and then find the path to t using one of the two neighbors of t' on ℓ. Let v denote a neighbor of t' on ℓ. Finally, we take the shortest of all $O(\frac{1}{\epsilon}\log n)$ candidate paths.

In order to find the subset of split lines we use the search tree $T_{\mathbf{u}}$ over the set of all split lines for a direction $\mathbf{u} \in \mathsf{N}$. For a node $z \in T_{\mathbf{u}}$, if t lies in the region left of split line at z we search the left child, else we search the right child. Searching $T_{\mathbf{u}}$ in this way, we reach a leaf node such that the associated region contains exactly one obstacle vertex w and the query point t. In this case we add a new split line ℓ^* between w and t and add Steiner vertices for the obstacle vertex w on ℓ^*. This gives us a total of $O(\log n) + 1$ split lines per direction that we need to search.

To compute the candidate paths, for a given a split line ℓ, we consider the Steiner vertices – projection t' and bypass t_1, t_2 – for the query point t. The shortest path from ℓ to t may either be $t' \to t$ or $t_2 \to t_1 \to t$. We find a neighbor v of t' or t_2 on ℓ (at most two neighbors are possible). We now consider the section of the path π_{vt} from v to t. If the arrays $dist_v, pred_v$ are not precomputed, which can happen if v is the projection of an obstacle vertex w on the new split line ℓ^*, we set $v = w$ and include the path from w to t along the split line ℓ^* to π_{vt}. (See also Figure 5.)

At this point we have found a vertex v such that $dist_v, pred_v$ are precomputed for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. Since the cost of bypass edges is zero, and all other segments in the path π_{vt} are axis-aligned, we can compute the cost $c(\pi_{vt})$ using the segment tree $S_{\mathbf{u}}$. The remaining cost budget is $h - c(\pi_{vt})$ which we *round up* for lookup in the $dist_v, pred_v$ arrays. More precisely, let j be the smallest integer such that $h - c(\pi_{vt}) \le j\epsilon$, then we compute the length of the candidate s–t path via v as $dist_v(j) + \|\pi_{vt}\|_1$. Finally, we take the minimum over all $O(\frac{1}{\epsilon}\log n)$ choices of v to obtain the shortest path π_{st} using the *pred* arrays. Using a similar argument as in the proof of Theorem 3, one can show that the length of π_{st} is at most $(1+\epsilon)$ times optimal and the cost is $(1+\epsilon)C$. The total query time is $O(\frac{1}{\epsilon}\log n \cdot \log n) = O(\frac{1}{\epsilon}\log^2 n)$.

Instead of computing the path itself, one may ask to just find the length of the shortest s–t path of cost at most C for some query point t. We can answer such queries approximately in $O(\frac{1}{\epsilon}\log^2 n)$ time using $O(\frac{n}{\epsilon^2}\log n \log \frac{h}{\epsilon})$ space. The main idea is that instead of storing

$O(\frac{h}{\epsilon})$ distance values in $dist_v$ for cost $0, \epsilon, 2\epsilon, \ldots, \frac{h}{\epsilon}\epsilon$, we store a subset of $O(\frac{1}{\epsilon}\log\frac{h}{\epsilon})$ values. More precisely, we only store the distance values corresponding to the cost $j\epsilon$ where j is the smallest integer such that $\epsilon(1+\epsilon)^i \leq j\epsilon$, for all i in $0, 1, 2, \ldots, \log_{1+\epsilon}\frac{h}{\epsilon}$. The size of $dist_v$ arrays for each vertex v is therefore $O(\log_{1+\epsilon}\frac{h}{\epsilon}) = O(\frac{1}{\epsilon}\log\frac{h}{\epsilon})$. Let π_{vt} be the path from v to t. The length of a s–t path via v has length $dist_v(i) + \|\pi_{vt}\|_1$, where i is the smallest integer with $h - c(\pi_{vt}) \leq \epsilon(1+\epsilon)^i$. Finally we take the minimum over all $O(\frac{1}{\epsilon}\log n)$ choices of v to obtain the shortest path π_{st}. We can show that the cost of π_{st} is at most $(1+5\epsilon)C$. Constructing the data structure for $\epsilon \leftarrow \epsilon/5$ we obtain the following theorem.

▶ **Theorem 10.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices,* s *be an obstacle vertex, and* $C \in \mathbb{R}$ *be a parameter. A data structure of* $O(\frac{nh}{\epsilon^2}\log n)$ *size can be constructed in* $O(\frac{nh}{\epsilon^2}\log n\log\frac{n}{\epsilon})$ *time such that, given a query point* $t \in \mathbb{R}^2$*, a path* π_{st} *can be returned with cost* $(1+\epsilon)C$ *and length at most* $(1+\epsilon)$ *times the optimal in* $O\big(\frac{1}{\epsilon}\log^2 n + k_{st}\big)$ *time, where* k_{st} *is the number of edges of* π_{st}*. The length of the path* π_{st} *can be returned in time* $O\big(\frac{1}{\epsilon}\log^2 n\big)$ *using a data structure of size* $O(\frac{n}{\epsilon^2}\log n\log\frac{h}{\epsilon})$*.*

Two point queries. Now we briefly explain how to extend the above data structure to handle two point queries. That is, both s, t are part of the query. During the preprocessing, we store distance values $dist_{uv}$ (similarly $pred_{uv}$) for every pair of vertices u, v in H_ϵ for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. The idea now is to find the neighbor u of s on some split line ℓ_s and neighbor v of t on split line ℓ_t. We compute the cost of paths π_{sv} and π_{vt} as before and set the length of this candidate s–t path to be $dist_{uv}(j) + \|\pi_{su}\|_1 + \|\pi_{vt}\|_1$. Here j is the smallest integer such that $h - c(\pi_{su}) - c(\pi_{vt}) \leq j\epsilon$. We take the minimum across $O(\frac{1}{\epsilon^2}\log^2 n)$ choices of u and v.

▶ **Theorem 11.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, and* $C \in \mathbb{R}$ *be a parameter. A data structure of* $O(\frac{n^2h}{\epsilon^3}\log^2 n)$ *size can be constructed in* $O(\frac{n^2h}{\epsilon^3}\log^2 n\log\frac{n}{\epsilon})$ *time such that, given two query points* $s, t \in \mathbb{R}^2$*, a path* π_{st} *can be returned with cost at most* $(1+\epsilon)C$ *and length at most* $(1+\epsilon)$ *times the optimal in* $O(\frac{1}{\epsilon^2}\log^2 n + k_{st})$ *time, where* k_{st} *is the number of edges of* π_{st}*. The length of the path* π_{st} *can be returned in* $O(\frac{1}{\epsilon^2}\log^2 n)$ *time using a data structure of size* $O(\frac{n^2}{\epsilon^3}\log^2 n\log\frac{h}{\epsilon})$*.*

6 Stochastic Shortest Path

In this section, we consider a stochastic model of obstacles where the existence of each obstacle $P_i \in \mathsf{P}$ is an independent event with known probability β_i. That is, P_i is part of the input with probability β_i and is not part of the input with probability $1-\beta_i$. We define the probability of path π_{st} as $\prod_{P_i\in S}(1-\beta_i)$ where $S \subseteq \mathsf{P}$ is the set of obstacles that this path goes through (assuming they did not exist). In such a setting, our goal is to compute the approximate shortest path that has probability more than a given threshold $\beta \in (e^{-1}, 1]$.

Let L_β denote the length of the shortest path from s to t with probability at least β. We convert the multiplicative costs to additive costs by setting $c_i = -\ln(1-\beta_i)$ for each obstacle and setting $C = -\ln\beta$. Using Theorem 9, we find a path π_{st} with length $L(\pi_{st}) \leq (1+\epsilon)L_\beta$ and cost $c(\pi_{st}) \leq (1+\epsilon)C$. It can be shown that π_{st} has probability at least $(1-\epsilon)\beta$.

▶ **Theorem 12.** *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, where each obstacle* $P_i \in \mathsf{P}$ *exists independently with a probability* β_i*,* s, t *be two obstacle vertices and* $\beta \in (e^{-1}, 1]$ *be a parameter. If* L_β *is the length of the shortest* s–t *path with probability at least* β*, a* s–t *path with length at most* $(1+\epsilon)L_\beta$ *and probability at least* $(1-\epsilon)\beta$ *can be computed in* $O(\frac{nh}{\epsilon^2}\log n\log\frac{n}{\epsilon})$ *time.*

Most likely path. We now consider the following question – given a bound L on the length of the path, what is the s–t path with maximum probability? We need a bound on the path length or else there is always a path of probability 1. To answer this question, we can again take negative logarithms of probabilities to transform into an additive cost model and construct the graph H_ϵ as before. Now instead of applying Theorem 3 on H_ϵ, we construct a new graph H_ϵ^* that is exactly the same as H_ϵ, but with length and cost parameters on edges interchanged. More precisely, for an edge $e \in H_\epsilon$ with length l_e and cost c_e, we have an edge $e^* \in H_\epsilon^*$ with length c_e and cost l_e. Next we apply Theorem 3 on the graph H_ϵ^* with $C = (1+\epsilon)L$, and scale all costs with a parameter $O(\frac{n}{C\epsilon}\log n)$, such that the target cost is scaled to $O(\frac{n}{\epsilon}\log n)$. We choose this value because a shortest path in H_ϵ can have $O(\frac{n}{\epsilon}\log n)$ edges. This gives us the following result.

▶ **Theorem 13.** *Let* P *be a set of* h *convex obstacles with* n *vertices,* s, t *be two obstacle vertices, and* $L \in \mathbb{R}$ *be a parameter. If* β_M *is the maximum probability of a path from* s *to* t *with length at most* L*, a path* π_{st} *with length at most* $(1+\epsilon)L$ *and probability at least* β_M *can be computed in* $O(\frac{n^2}{\epsilon^3}\log^2 n \log\frac{n}{\epsilon})$ *time.*

References

1 Esther M Arkin, Joseph SB Mitchell, and Christine D Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom*, pages 153–156, 1991.

2 T. M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005.

3 T. M. Chan and Y. Nekrich. Towards an optimal method for dynamic planar point location. In *Proc. 56th Symp. Found. Comp. Science*, pages 390–409. IEEE, 2015.

4 D. Z. Chen, K. S. Klenk, and H. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM J. Comput.*, 29(4):1223–1246, 2000.

5 Danny Z Chen, Ovidiu Daescu, and Kevin S Klenk. On geometric path query problems. *Int. J. Comp. Geom. & Applic.*, 11(06):617–645, 2001.

6 Danny Z Chen, John Hershberger, and Haitao Wang. Computing shortest paths amid convex pseudodisks. *SIAM J. Comput.*, 42(3):1158–1184, 2013.

7 Danny Z Chen, Rajasekhar Inkulu, and Haitao Wang. Two-point L_1 shortest path queries in the plane. In *Proc. 30th Annual Symp. Comput. Geom.*, page 406. ACM, 2014.

8 Danny Z Chen and Haitao Wang. A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In *Proc. 19th Europ. Symp. Alg.*, pages 481–492. Springer, 2011.

9 Danny Z Chen and Haitao Wang. L_1 shortest path queries among polygonal obstacles in the plane. In *Proc. 30th Int. Symp. Theor. Asp. Comp. Science*, volume 20, 2013.

10 Danny Z Chen and Haitao Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11(4):26, 2015.

11 Danny Z Chen and Haitao Wang. A new algorithm for computing visibility graphs of polygonal obstacles in the plane. *J. Comput. Geom.*, 6(1):316–345, 2015.

12 Y.J Chiang and J.S.B Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proc. 10th ACM-SIAM Annual Symp. Discrete Algorithms*. SIAM, 1999.

13 K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n\log^2 n)$ time. In *Proc. 3rd Annual Symp. Comput. Geom.*, pages 251–257. ACM, 1987.

14 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *J. Algorithms*, 62(1):1–18, 2007.

15 Y. Gao. Shortest path problem with uncertain arc lengths. *Computers & Mathematics with Applications*, 62(6):2591–2600, 2011.

16 Subir Kumar Ghosh. *Visibility algorithms in the plane*. Cambridge university press, 2007.

17 S. Har-Peled and V. Koltun. Separability with outliers. In *Proc. Int. Symp. Alg. and Comput.*, pages 28–39. Springer, 2005.

18 J. Hershberger, N. Kumar, and S. Suri. Shortest paths in the plane with obstacle violations. In *Proc. 25th Annual Eur. Symp. on Alg.*, volume 87, pages 49:1–49:14, 2017.

19 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

20 Rajasekhar Inkulu and Sanjiv Kapoor. Planar rectilinear shortest path computation using corridors. *J. Comput. Geom.*, 42(9):873–884, 2009.

21 M Iwai, H Suzuki, and T Nishizeki. Shortest path algorithm in the plane with rectilinear polygonal obstacles. In *Proc. SIGAL Workshop*, 1994.

22 P. Kamousi, T M Chan, and S. Suri. Stochastic minimum spanning trees in Euclidean spaces. In *Proc. 27th Annual Symp. Comput. Geom.*, pages 65–74. ACM, 2011.

23 D.T Lee, C.-D. Yang, and T.H Chen. Shortest rectilinear paths among weighted obstacle. *Int. J. Comput. Geom. & Appl.*, 1(02):109–124, 1991.

24 D.T Lee, C.D Yang, and C.K. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70(3):185–215, 1996.

25 A. Maheshwari, S. C. Nandy, D. Pattanayak, S. Roy, and M. Smid. Geometric path problems with violations. *Algorithmica*, pages 1–24, 2016.

26 J. Matoušek. On geometric optimization with few violated constraints. *Discrete & Computational Geometry*, 14(4):365–384, 1995.

27 J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

28 Joseph S.B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, 1998.

29 Evdokia Nikolova, Matthew Brand, and David R Karger. Optimal route planning under uncertainty. In *Proc. 16th Int. Conf. Autom. Plann. and Sched.*, volume 6, pages 131–141, 2006.

30 T. Roos and P. Widmayer. k-violation linear programming. *Inf. Process. Lett.*, 52(2):109–114, 1994.

31 Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

32 Neil Sarnak and Robert E Tarjan. Planar point location using persistent search trees. *Communic. ACM*, 29(7):669–679, 1986.

33 H. Wang. Bicriteria rectilinear shortest paths among rectilinear obstacles in the plane. In *Proc. 33rd Annual Symp. Comput. Geom.*, pages 60:1–60:16, 2017.

34 C.D Yang, D.T. Lee, and C.K Wong. On bends and lengths of rectilinear paths: a graph-theoretic approach. *Int. J. Comput. Geom. & Appl.*, 2(01):61–74, 1992.

35 C.D Yang, D.T. Lee, and C.K. Wong. Rectilinear path problems among rectilinear obstacles revisited. *SIAM J. Comput.*, 24(3):457–472, 1995.

On Romeo and Juliet Problems: Minimizing Distance-to-Sight

Hee-Kap Ahn
Department of Computer Science and Engineering, POSTECH
Pohang, South Korea
heekap@postech.ac.kr

Eunjin Oh
Department of Computer Science and Engineering, POSTECH
Pohang, South Korea
jin9082@postech.ac.kr

Lena Schlipf
Theoretische Informatik, FernUniversität in Hagen
Hagen, Germany
lena.schlipf@fernuni-hagen.de

Fabian Stehn
Institut für Informatik, Universität Bayreuth
Bayreuth, Germany
fabian.stehn@uni-bayreuth.de

Darren Strash
Department of Computer Science, Colgate University
Hamilton, USA
dstrash@cs.colgate.edu
https://orcid.org/0000-0001-7095-8749

Abstract

We introduce a variant of the watchman route problem, which we call the *quickest pair-visibility* problem. Given two persons standing at points s and t in a simple polygon P with no holes, we want to minimize the distance these persons travel in order to see each other in P. We solve two variants of this problem, one minimizing the longer distance the two persons travel (min-max) and one minimizing the total travel distance (min-sum), optimally in linear time. We also consider a query version of this problem for the min-max variant. We can preprocess a simple n-gon in linear time so that the minimum of the longer distance the two persons travel can be computed in $O(\log^2 n)$ time for any two query positions where the two persons lie.

2012 ACM Subject Classification Theory of computation → Computational geometry, Mathematics of computing → Paths and connectivity problems

Keywords and phrases Visibility polygon, shortest-path, watchman problems

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.6

Funding This work by Ahn and Oh was supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00905) supervised by the IITP (Institute for Information & Communications Technology Promotion).

Acknowledgements This research was initiated at the 19th Korean Workshop on Computational Geometry in Würzburg, Germany.

1 Introduction

In the watchman route problem, a watchman takes a route to *guard* a given region—that is, any point in the region is visible from at least one point on the route. It is desirable to make the route as short as possible so that the entire area can be guarded as quickly as possible. The problem was first introduced in 1986 by Chin and Ntafos [4] and has been extensively studied in computational geometry [3, 14]. Though the problem is NP-hard for polygons with holes [4, 5, 7], an optimal route can be computed in time $O(n^3 \log n)$ for simple n-gons [6] when the tour must pass through a specified point, and $O(n^4 \log n)$ time otherwise.

In this paper, we study a variant we call the *quickest pair-visibility* problem, which can be stated as follows.

▶ **Problem** (quickest pair-visibility problem)**.** *Given two points s and t in a simple polygon P, compute the minimum distance that s and t must travel in order to see each other in P.*

This problem may sound similar to the shortest path problem between s and t, in which the objective is to compute the shortest path for s to *reach* t. However, they differ even for a simple case: for any two points lying in a convex polygon, the distance in the quickest pair-visibility problem is zero while in the shortest path problem it is their Euclidean distance.

The quickest pair-visibility problem occurs in optimization tasks. For example, mobile robots that use a line-of-sight communication model are required to move to mutually-visible positions to establish communication [8]. An optimization task here is to find shortest paths for the robots to meet the visibility requirement for establishing communication among them.

Wynters and Mitchell [16] studied this problem for two agents acting in a polygonal domain in the presence of polygonal obstacles and gave an $O(nm)$-time algorithm for the min-sum variant (where m is the number of edges of the visibility graph of all corners) and an $O(n^3 \log n)$-time algorithm for the min-max variant. A query version of the quickest visibility problem has also been studied [1, 13, 15]. In the query problem, a polygon and a source point lying in the polygon are given, and the goal is to preprocess them and construct a data structure that allows, for a given query point, to find the shortest path taken from the source point to see the query point efficiently. Khosravi and Ghodsi [13] considered the case for a simple n-gon and presented an algorithm to construct a data structure of $O(n^2)$ space so that given a query, it finds the shortest visibility path in $O(\log n)$ time. Later, Arkin et al. [1] improved the result and presented an algorithm for the problem in a polygonal domain. Very recently, Wang [15] presented an improved algorithm for this problem for the case that the number of the holes in the polygon is relatively small. Figure 1(a) illustrates differences in these problems for a simple polygon and two points, s and t, in the polygon.

1.1 Our results

In this paper, we consider two variants of the quickest pair-visibility problem for a simple polygon: either we want to minimize the maximum length of a traveled path (*min-max variant*) or we want to minimize the sum of the lengths of both traveled paths (*min-sum variant*) We give a sweep-line-like approach that "rotates" the lines-of-sight along vertices on the shortest path between the start positions, allowing us to evaluate a linear number of candidate solutions on these lines. Throughout the sweep, we encounter solutions to both variants of the problem. We further show that our technique can be implemented in linear time.

We also consider a query version of this problem for the min-max variant. We can preprocess a simple n-gon in linear time so that the minimum of the longer distance the two query points travel can be computed in $O(\log^2 n)$ time for any two query points.

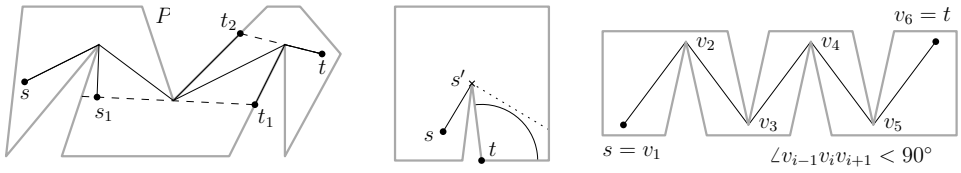


Figure 1 (a) The quickest pair-visibility problem finds two paths $\pi(s, s_1)$ and $\pi(t, t_1)$ such that $\overline{s_1t_1} \subset P$ and $\max\{|\pi(s, s_1)|, |\pi(t, t_1)|\}$ or $|\pi(s, s_1)| + |\pi(t, t_1)|$ is minimized. The quickest visibility problem for query point t finds a shortest $\pi(s, t_2)$ with $\overline{tt_2} \subset P$. (b) **min-max:** Every pair (s', t^*), where t^* is some point within the geodesic disk centered in t with radius $\pi(s, s')$, is an optimal solution to the *min-max* problem. (c) **min-sum:** Every pair (v_i, v_{i+1}) for $1 \leq i < 6$ is an optimal solution to this instance.

2 Preliminaries

Let P be a simple polygon and ∂P be its boundary. The vertices of P are given in counterclockwise order along ∂P. We denote the shortest path within P between two points $p, q \in P$ by $\pi(p, q)$ and its length by $|\pi(p, q)|$. Likewise, we denote the shortest path within P between a point $p \in P$ and a line segment $\ell \in P$ by $\pi(p, \ell)$. We say a point $p \in P$ is visible from another point $q \in P$ (and q is visible from p) if and only if line segment $\overline{pq}$ is completely contained in P.

For two starting points s and t, our task is to compute a pair (s', t') of points such that s' and t' are visible to each other, where we wish to minimize the lengths of $\pi(s, s')$, and $\pi(t, t')$. In the *min-max* setting, we wish to minimize $\max\{|\pi(s, s')|, |\pi(t, t')|\}$. For the *min-sum* setting, we wish to minimize $|\pi(s, s')| + |\pi(t, t')|$. Note that, for both variants, the optimum is not necessarily unique; see Figure 1(b) and (c).

For our discussion, let (s^*, t^*) be an optimal solution for the instance at hand. Let $V(p)$ denote the visible region for a point p in P, that is, the portion of P that is visible from p. Clearly, $V(p)$ is a *star-shaped* polygon. Moreover, every boundary edge of $V(p)$ is either (part of) an edge of P or a segment $\overline{vq}$ that is contained in P and parallel to $\overline{pv}$, where v is a vertex of P visible from p and q is a point on the boundary of P. We call an edge of the latter type a *window edge* of the visibility region. The structure of $V(p)$ may change as p moves along a path contained in P. It is known that a change to the structure of $V(p)$ occurs if and only if two vertices of P become collinear with p [2].

We say a segment g is tangent to a path π at a vertex v if $v \in g \cap \pi$ and v's neighboring vertices on π are on the same side of g.

▶ **Lemma 1.** *Unless s and t are visible to each other, the segment $\overline{s^*t^*}$ is tangent to the shortest path $\pi(s, t)$ at a vertex v of $\pi(s, t)$.*

Proof. We first show that there is a vertex of P lying on $\overline{s^*t^*}$. Consider the visibility regions $V(s^*)$ and $V(t^*)$. If s^* lies on a window edge e of $V(t^*)$, then e has a vertex v of P as its endpoint closer to t^*. Therefore v lies on $\overline{s^*t^*}$. The case that t^* lies on a window edge of $V(s^*)$ can be shown similarly. So assume that this is not the case, that is, s^* is in $V(t^*)$ but not in a window edge of $V(t^*)$ and t^* is in $V(s^*)$ but not in a window edge of $V(s^*)$. Then there is a point s' on $\pi(s, s^*) \cap V(t^*)$ infinitesimally close to s^* and a point t' on $\pi(t, t^*) \cap V(s^*)$ infinitesimally close to t^* such that $|\pi(s, s')| < |\pi(s, s^*)|$ and $|\pi(t, t')| < |\pi(t, t^*)|$, and s' and t' still see each other. This contradicts the optimality of (s^*, t^*).

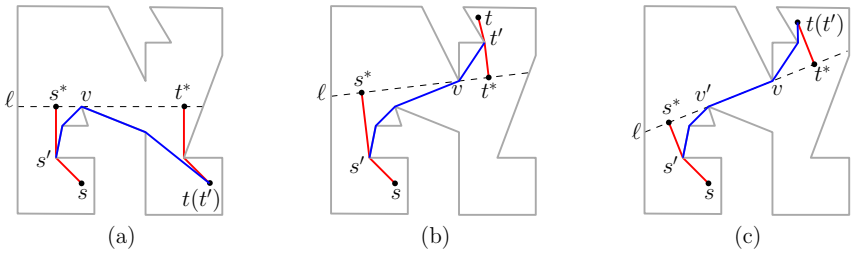


Figure 2 (a) Both s and t lie on the same side of ℓ through s^* and t^*. (b) If there is only one vertex v of P lying on $\overline{s^*t^*}$, we can always find another optimal pair of points by rotating ℓ around v and taking the closest points from s and t on the rotated ℓ under the geodesic metric. (c) The shortest path $\pi(s,t)$ passes through v and v'.

We now show that $\overline{s^*t^*}$ contains a vertex of $\pi(s,t)$. Let s' be the last vertex that $\pi(s,t)$ and $\pi(s,s^*)$ have in common from s, and let t' be the last vertex that $\pi(s,t)$ and $\pi(t,t^*)$ have in common from t. Since a subpath of a shortest path is also shortest, $\pi(s',t')$ is the subpath of $\pi(s,t)$ from s' to t'. Assume to the contrary that $\pi(s',t')$ (and therefore $\pi(s,t)$) contains no vertex of P that is also on $\overline{s^*t^*}$. There are two cases: (a) both s and t lie on the same side of the line ℓ through s^* and t^*, or (b) s and t lie on different sides of ℓ.

For case (a), ∂P touches ℓ at a vertex v lying between s^* and t^* locally from the same side of ℓ that s and t lie. Otherwise, (s^*,t^*) is not optimal as both $\pi(s,\ell)$ and $\pi(t,\ell)$ become shorter by a rotation of ℓ. See Figure 2(a). Consider the portion (line segment) of ℓ visible from v, which is split into two segments by v, one containing s^* and one containing t^*. Since $\pi(s',t')$ does not contain v, it has to cross the segment containing s^*. But then it must cross ℓ again in the segment containing t^* to reach t'. Since the portion of the path between the two crossing points can be shortened by the segment connecting them, which contradicts to the assumption that $\pi(s',t')$ is a shortest path. Thus v lies on $\pi(s,t)$ and $\overline{s^*t^*}$ is tangent to $\pi(s,t)$ at v.

For case (b), without loss of generality, assume that s lies below ℓ and t lies above ℓ. If there is only one vertex v of P lying on $\overline{s^*t^*}$, we can always find another pair of points $(\hat{s},\hat{t})$ such that $\hat{s}$ and $\hat{t}$ are visible to each other and they satisfy either (1) $|\pi(s,\hat{s})| < |\pi(s,s^*)|$ and $|\pi(t,\hat{t})| \leq |\pi(t,t^*)|$ or (2) $|\pi(s,\hat{s})| \leq |\pi(s,s^*)|$ and $|\pi(t,\hat{t})| < |\pi(t,t^*)|$. (The equality holds if s^* or t^* coincides with v.) Such points $\hat{s}$ and $\hat{t}$ can be obtained by rotating ℓ around v and taking the closest points from s and t, respectively, on the rotated ℓ under the geodesic metric. See Figure 2(b) for an illustration. Therefore we assume that there are two vertices v and v' that touch $\overline{s^*t^*}$ from above and from below, respectively. Since s lies below ℓ, v' comes before v from s^* along $\overline{s^*t^*}$ to t^*. See Figure 2(c). Consider the portion (line segment) of ℓ visible from v, which is split into two segments by v, one contains s^* and one contains t^*. If $\pi(s',t')$ crosses the segment containing s^*, it must cross ℓ again in the segment containing t^* to reach t'. Then the path between the two crossing points can be shortened by the segment connecting them, which is a contradiction. Thus, $\pi(s',t')$ passes through v'. The proofs for v and v' are symmetric, and thus both vertices v and v' are on the shortest path $\pi(s,t)$ which in turn also establishes, that $\overline{s^*t^*}$ is (locally) tangent to $\pi(s,t)$ at v and at v'. ◀

3 Computing All Events for a Sweep-Line-Like Approach

For each vertex v on $\pi(s,t)$ we compute a finite collection of lines through v, each being a configuration at which the combinatorial structure of the shortest paths $\pi(s,s^*)$ and/or $\pi(t,t^*)$ changes. To be more precise, at these lines either the vertices of $\pi(s,s^*)$ or $\pi(t,t^*)$

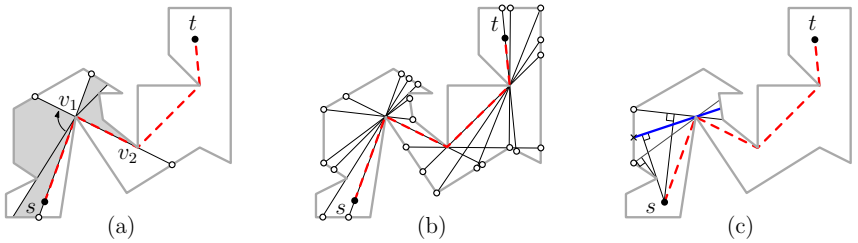


Figure 3 Path-, boundary-, and bend-events. (a) The endpoints of the line-of-sight through $\overline{sv_1}$ make up the first path-event. The line-of-sight rotates until it hits the next path-event: the endpoints of the line-of-sight through $\overline{v_1v_2}$. (b) All path- and boundary-events: the event-queue is initialized with these events. (c) A bend-event (marked with a cross) occurs between the two boundary-events. The shortest path from s to these segments changes at the bend-event.

(except for s^* and t^*) change or the edge of ∂P changes that is intersected by the extension of $\overline{s^*t^*}$. Notice that in the remaining part of the paper (s^*, t^*) is the optimal solution pair from s, t to the a given line (and not necessarily a global optimal solution for the quickest pair-visibility problem). To explain how to compute these lines, we introduce the concept of a *line-of-sight*.

▶ **Definition 2** (line-of-sight)**.** We call a segment ℓ a *line-of-sight* if (i) $\ell \subset P$, (ii) both endpoints of ℓ lie on ∂P, and (iii) ℓ is tangent to $\pi(s,t)$ at a vertex $v \in \pi(s,t)$.

The algorithm we present is in many aspects similar to a sweep-line strategy, except that we do not sweep over the scene in a standard fashion but rotate a *line-of-sight* ℓ in P around the vertices of the shortest path $\pi(s,t) := (s = v_0), v_1, \ldots, v_{k-1}, (t = v_k)$, making use of Lemma 1. The process will be initialized with a line-of-sight that contains s and v_1 and is then rotated around v_1 (while remaining tangent to v_1) until it hits v_2, see Figure 3(a). In general, the current line-of-sight is rotated around v_i in a way so that it remains tangent to v_i (it is rotated in the interior of P) until the line-of-sight contains v_i and v_{i+1}, then the process is iterated with v_{i+1} as the new rotation center. The process terminates as soon as the line-of-sight contains v_{k-1} and t.

While performing these rotations around the shortest path vertices, we encounter all combinatorially different lines-of-sight. As for a standard sweep-line approach, we will compute and consider events at which the structure of a solution changes: this is either because the interior vertices of $\pi(s,s^*)$ or $\pi(t,t^*)$ change or because the line-of-sight starts or ends at a different edge of ∂P. These events will be represented by points on ∂P (actually, we introduce the events as vertices on ∂P unless they are already vertices). Between two consecutive lines-of-sight, we compute the local minima of the relevant distances for the variant at hand in constant time and hence encounter all global minima eventually.

There are three event-types to distinguish:

1. **Path-Events** are endpoints of lines-of-sight that contain two consecutive vertices of the shortest path $\pi(s,t)$. See Figure 3(a).
2. **Boundary-Events** are endpoints of lines-of-sight that are tangent at a vertex of $\pi(s,t)$ and contain at least one vertex of $P \setminus \pi(s,t)$ (potentially as an endpoint). See Figure 3(b).
3. **Bend-Events** are endpoints of lines-of-sight where the shortest path from s (or t) to the line-of-sight gains or loses a vertex while rotating the line-of-sight around a vertex v. See Figure 3(c). Note that bend-events can coincide with path- or boundary-events.

We will need to explicitly know both endpoints of the line-of-sight on ∂P at each event and the corresponding vertex of $\pi(s,t)$ on which we rotate.

▶ **Lemma 3** (Computing path- and boundary-events). *For a simple polygon P with n vertices and points $s, t \in P$, the queue $\mathcal{Q}$ of all path- and boundary-events of the rotational sweep process, ordered according to the sequence in which the sweeping line-of-sight encounters them, can be initialized in $O(n)$ time.*

Proof. Consider some line-of-sight ℓ that is tangent to a vertex $v_i \in \pi(s,t)$ for some $0 < i < k$. Then ℓ subdivides P into a number of subpolygons. Consider ℓ as the union of two (sub)segments ℓ^+ and ℓ^- of ℓ induced by v_i such that $\ell^+ \cap \ell^- = \{v_i\}$ and ℓ^- is incident to the subpolygon of P induced by ℓ containing s.

We will discuss the computation of all boundary- and path-events swept by ℓ^+. The other events swept by ℓ^- can be computed in a second round by changing the roles of s and t. We do not maintain a queue for the events explicitly; instead we will introduce new vertices on ∂P or label existing vertices of ∂P as events. Later the events will be considered by following two pointers to vertices on ∂P and hence by processing the vertices in the order that they appear on ∂P.

We start with computing all path-events swept by ℓ^+. For this we compute the shortest path map M_s of s in P. The shortest path map of s is a decomposition of P in $O(n)$ triangular cells such that the shortest path from s to any point within a cell is combinatorially the same. It can be obtained by extending every edge of the shortest path tree of s towards its descendants until it reaches ∂P in linear time [10]. A path-event occurs when a line-of-sight contains two consecutive vertices of $\pi(s,t)$. Note that for each path-event, ℓ^+ appears as an edge of M_s and its endpoints appear as vertices of M_s. For each index i with $0 < i \leq k$, we find the edge incident to v_i and parallel to $\overline{v_{i-1}v_i}$ by considering every edge of M_s incident to v_i. This takes $O(n)$ time in total since there are $O(n)$ edges of M_s and we consider every edge at most once.

For computing the boundary-events, we use the following properties. While rotating around v_i from the position where ℓ contains v_{i-1} to the position in which ℓ contains v_{i+1}, let A_i^+ (A_i^-) be the region of P that is swept over by ℓ^+ (ℓ^-). (See Figure 4.) Observe that
P1 all A_i^+ for $1 < i < k$ are pairwise disjoint,
P2 all A_i^- for $1 < i < k$ are pairwise disjoint,
P3 for all $1 < i < k$ and all points $p \in A_i^+$ the shortest path $\pi(p,s)$ contains v_i,
P4 for all $1 < i < k$ and all points $p \in A_i^-$ the shortest path $\pi(p,t)$ contains v_i.

To compute all boundary-events that are vertices of P swept by ℓ^+, we will make use of the shortest path tree T_s for s in P. A boundary-event x is defined by a vertex $v_i \in \pi(s,t)$ such that the line-of-sight that contains x (potentially as one endpoint) is tangent to $\pi(s,t)$ in v_i. It follows from Property P3, that $\overline{v_i x}$ is an edge of T_s (and by that it cannot be obstructed by other edges of P) and $x \notin \pi(s,t)$. So the vertices of P whose parent vertex in T_s is a vertex of $\pi(s,t)$ are possible boundary-events. In order to compute all boundary-events we consider all consecutive path-events and compute all corresponding boundary-events by following ∂P and checking the vertices within the candidate set. We compute the boundary-events which are vertices of P swept by ℓ^- in a similar way.

So far we labeled all vertices x on ∂P that are boundary-events. We still need to compute the other endpoint $\tilde{x}$ of the line-of-sight $\overline{x\tilde{x}}$ that is tangent in v_i. Let $\overline{x_i\tilde{x}_i}$ be the line-of-sight at the path-event x_i so that $\tilde{x}_i, v_{i-1}, v_i, x_i \in \ell$. (See Figure 4.) While rotating ℓ around v_i, ℓ^+ sweeps over A_i^+ until the next path-event is met. Let E_i^+ be the sequence of the path- and boundary-events in A_i^+ we obtained so far sorted in counter-clockwise order along ∂P. The order of events in E_i^+ is the same as the order in which ℓ^+ sweeps over them. Our goal is to compute $\tilde{x}$ for every event in E_i^+ in order. To do this, we consider the (triangular) cells

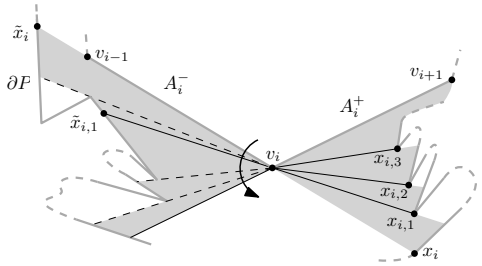


Figure 4 Let $E_i^+ = \langle x_{i,1}, \ldots, x_{i,k} \rangle$ for an index $1 \leq k \leq n$. We start at $\tilde{x}_i$ and follow the (triangular) cells of M_t incident to v_i in counter-clockwise order around v_i until we find $\tilde{x}_{i,1}$. Then we continue to follow such cells until we find $\tilde{x}_{i,2}$, and so on.

of M_t incident to v_i one by one in counter-clockwise order around v_i starting from the cell incident to $\tilde{x}_i$. Since every point in such cells is visible from v_i, we can determine if $\tilde{x}$ is contained in a cell in constant time for any event $x \in E_i^+$. Therefore, we can compute $\tilde{x}$ for every event x in E_i^+ in time linear in the number of the cells of M_t incident to v_i and the number of events of E_i^+, giving us all path- and boundary-events in $O(n)$ total time. ◀

Once we initialized the event queue $\mathcal{Q}$, we can now compute and process bend-events as we proceed in our line-of-sight rotations.

▶ **Lemma 4.** *All bend-events can be computed in $O(n)$ time, sorted in the order as they appear on the boundary of P.*

Proof. We assume that all path- and boundary-events are already computed. Additionally, we assume that all vertices of the boundary- and path-events (the endpoints of the corresponding line-of-sights) are inserted on ∂P. Recall that, for each event, we know both endpoints of the line-of-sight ℓ on ∂P and the corresponding vertex of $\pi(s,t)$ on which we rotate.

As in the proof of Lemma 3, we consider the line-of-sight ℓ tangent to a vertex $v \in \pi(s,t)$ as the union of two (sub)segments ℓ^+ and ℓ^- of ℓ induced by v such that $\ell^+ \cap \ell^- = \{v\}$ and ℓ^- is incident to the subpolygon of P induced by ℓ containing s. We discuss the computation of all bend-events that are encountered by ℓ^-. The bend-events that are swept over by ℓ^+ can be computed in a second round by changing the roles of s and t.

We start with the path-event defined by s and v_1, and consider all events in the order they appear. Let ℓ be the current line-of-sight rotating around a vertex v and denote by x the endpoint of ℓ^- other than v. To find the bend-events efficiently, we compute and maintain the shortest path $\pi(s,\ell)$ over the events.

While ℓ rotates around v, the combinatorial structure of $\pi(s,\ell)$ may change. Specifically, let $e_\ell = (u,w)$ denote the edge of $\pi(s,\ell)$ incident to ℓ with w on ℓ. Note that during the rotation of ℓ, all the edges of $\pi(s,\ell)$ are stationary, except that e_ℓ rotates around u. Therefore, a change in the combinatorial structure of $\pi(s,\ell)$ occurs only when (1) e_ℓ hits a vertex u' of P and splits into two edges sharing u' or (2) the two edges of $\pi(s,\ell)$ incident to u become parallel. (Then they merge into one and u disappears from the shortest path.) See Figure 5. From any event of the two event types above, e_ℓ, u, and $\pi(s,\ell)$ are updated accordingly. Additionally, x is updated and its new position is inserted as vertex on ∂P as it represents a bend-event.

▶ **Lemma 5.** *An event of type (1) occurs only when (a) x reaches a vertex u', or (b) e_ℓ hits a vertex u' of $\pi(s,t)$ in its interior. Moreover, for case (b), u and u' are consecutive in $\pi(s,t)$.*

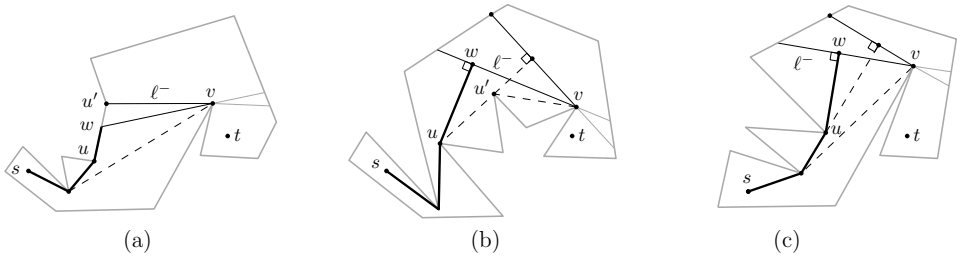


Figure 5 (a) A bend-event of type (1) occurs when $x = u_\ell$ reaches u'. (b) A bend-event of type (1) occurs when $e_\ell = \overline{uw}$ hits a vertex u' of $\pi(s,t)$. (c) A bend-event of type (2) occurs when two edges incident to u are parallel.

Proof. Consider the case that e_ℓ is not orthogonal to ℓ. Then the closest point in ℓ from s is x. Thus, the only way that e_ℓ hits a vertex of P is that x reaches u'. See Figure 5(a).

Now consider the case that e_ℓ is orthogonal to ℓ. Then u' is contained in $\pi(u,v)$. See Figure 5(b). Since $\pi(u,v)$ is a subpath of $\pi(s,t)$, u' is a vertex of $\pi(s,t)$, and thus u is the vertex of $\pi(s,t)$ previous to u' from s. ◀

▶ **Lemma 6.** *Once a vertex disappears from $\pi(s,\ell)$, it never appears again on the shortest path during the rotation of the current line-of-sight ℓ.*

Proof. Assume to the contrary that there is a vertex u that disappears from $\pi(s,\ell_1)$, but then appears again on $\pi(s,\ell_2)$ for two line-of-sights ℓ_1 and ℓ_2 during the rotation. Since both $\pi(s,\ell_1)$ and $\pi(s,\ell_2)$ contain u in its interior, both of them also contain $\pi(s,u)$. Since u disappears from $\pi(s,\ell_1)$, the edge of $\pi(s,\ell_1)$ incident to u is orthogonal to ℓ_1. We claim that u appears on $\pi(s,\ell_2)$ due to case (b) of type(1), that is, the edge of $\pi(s,\ell_2)$ incident to ℓ_2 hits u. Assume to the contrary that u appears on $\pi(s,\ell_2)$ due to case (a) of type (1). However, u (and its event vertex on ∂P) is already swept by a line-of-sight before we consider ℓ_2 because it appears on $\pi(s,\ell_1)$. Thus, u appears on $\pi(s,\ell_2)$ due to case (b) of (2), and the edge of $\pi(s,\ell_2)$ incident to u is orthogonal to ℓ_2. This means that ℓ_1 and ℓ_2 are parallel.

Since ℓ_1 and ℓ_2 are parallel, they are tangent to $\pi(s,t)$ at two distinct vertices, say v_1 and v_2, respectively. Moreover, the path $\pi(p_1,p_2)$ contains v_1 for any two points $p_1 \in P_1$ and $p_2 \in \ell_2$, where P_1 is the subpolygon bounded by ℓ_1^- containing s. Thus, $\pi(s,\ell_2)$ contains $\pi(s,v_1)$, and no vertex in P_1 other than the vertices of $\pi(s,v_1)$ appears on $\pi(s,\ell_2)$. Since u is contained in P_1, it cannot appear on $\pi(s,\ell_2)$, which is a contradiction. ◀

We can update u, e_ℓ, x and $\pi(s,\ell)$ in constant time for a type (1) event. We can update them in $O(n)$ time for all type (2) events in total by Lemma 6. The vertices representing the bend-events can be inserted on ∂P in the same time. ◀

4 Algorithm Based on a Sweep-Line-Like Approach

In this section, we present a linear-time algorithm for computing the minimum distance that two points s and t in a simple polygon P travel in order to see each order. We compute all events defined in Section 3 in linear time. The remaining task is to handle the lines-of-sight lying between two consecutive events.

▶ **Lemma 7.** *For any two consecutive events, the line-of-sight ℓ lying between them that minimizes the sum of the distances from s and t to ℓ can be found in constant time.*

Proof. Let $\mathcal{L}$ be the set of all lines-of-sight lying between the two consecutive events. Every line-of-sight in $\mathcal{L}$ contains a common vertex v of $\pi(s,t)$. We assume that $\mathcal{L}$ contains no vertical line-of-sight. Otherwise, we consider the set containing all lines-of-sight of $\mathcal{L}$ with positive slopes, and then the set containing all lines-of-sight of $\mathcal{L}$ with negative slopes.

By construction, the second to the last vertex u of $\pi(s,\ell)$ (and $\pi(t,\ell)$) for any $\ell \in \mathcal{L}$ remains the same. We already obtained v and u while computing the events. We will give an algebraic function for the length of $\pi(s,\ell)$ for $\ell \in \mathcal{L}$. An algebraic function for the length of $\pi(t,\ell)$ can be obtained by changing the roles of s and t.

Since the topology of $\pi(s,\ell)$ for every $\ell \in \mathcal{L}$ remains the same, we consider only the length of $\pi(u,\ell)$. Observe that $\pi(u,\ell)$ is a line segment for any $\ell \in \mathcal{L}$, and thus its length is the same as the Euclidean distance between u and ℓ. The length is either the Euclidean distance between u and the line containing ℓ, or the Euclidean distance between u and the endpoint of ℓ closest to u. We show how to handle the first case only because the second case can be handled analogously.

To use this observation, we use $\ell(\alpha)$ to denote the line of slope α passing through v for any $\alpha > 0$. There is an interval I such that $\ell(\alpha)$ contains a line-of-sight in $\mathcal{L}$ if and only if $\alpha \in I$. The Euclidean distance between u and $\ell(\alpha)$ is the same as the distance between u and the line-of-sight contained in $\ell(\alpha)$. Thus, in the following, we consider the distance between u and $\ell(\alpha)$ for every $\alpha \in I$.

Since $\ell(\alpha)$ passes through a common vertex, the line $\ell(\alpha)$ can be represented as the form of $y = \alpha x + f(\alpha)$, where $f(\alpha)$ is a function linear in α. Then, the distance between u and $\ell(\alpha)$ can be represented as the form of $|c_1\alpha + c_2|/\sqrt{\alpha^2+1}$, where c_1 and c_2 are constants depending only on v and u.

Then our problem reduces to the problem of finding a minimum of the function of the form of $(|c_1\alpha + c_2| + |c_1'\alpha + c_2'|)/\sqrt{\alpha^2+1}$ for four constants c_1, c_2, c_1' and c_2', and for all $\alpha \in I$. We can find a minimum in constant time using an elementary analysis. ◀

▶ **Lemma 8.** *For any two consecutive events, the line-of-sight ℓ lying between the them that minimizes the maximum of the distances from s and t to ℓ can be found in constant time.*

▶ **Theorem 9.** *Given a simple n-gon P with no holes and two points $s, t \in P$, a point-pair (s^*, t^*) such that* i) $\overline{s^*t^*} \subset P$ *and* ii) *either $|\pi(s,s^*)| + \pi(t,t^*)|$ or $\max\{|\pi(s,s^*)|, |\pi(t,t^*)|\}$ is minimized can be computed in $O(n)$ time.*

Proof. Our algorithm first computes all path- and boundary-events as described in Lemma 3. The number of events introduced during this phase is bounded by the number of vertices of the shortest path maps, M_s and M_t, respectively, which are $O(n)$. In the next step, it computes the bend-events on ∂P as described in Lemma 4, which can be done in $O(n)$ time. Finally, our algorithm traverses the sequence of events. Between any two consecutive events, it computes the respective local optimum in constant time by Lemma 7. It maintains the smallest one among the local optima computed so far, and return it once all events are processed. Therefore the running time of the algorithm is $O(n)$.

For the correctness, consider the combinatorial structure of a solution and how it changes. The path-events ensure that all vertices of $\pi(s,t)$ are considered as being the vertex lying on the segment connecting the solution (s^*, t^*). While the line-of-sight rotates around one fixed vertex of $\pi(s,t)$, either the endpoints of line-of-sight sweep over or become tangent to a vertex of ∂P. These are exactly the boundary-events. Or the combinatorial structure of $\pi(s,s^*)$ or $\pi(t,t^*)$ changes as interior vertices of $\pi(s,s^*)$ or $\pi(t,t^*)$ appear or disappear. These happen exactly at bend-events. Therefore, our algorithm returns an optimal point-pair. ◀

5 Quickest Pair-Visibility Query Problem

In this section, we consider a query version of the min-max variant of the quickest pair-visibility problem: Preprocess a simple n-gon P so that the minimum traveling distance for two query points s and t to see each other can be computed efficiently. We can preprocess a simple n-gon in linear time and answer a query in $O(\log^2 n)$ time by combining the approach in Section 4 with the data structure given by Guibas and Hershberger [9, 11]. For any two query points s and t in P, the query algorithm for their data structure returns $\pi(s,t)$ represented as a binary tree of height $O(\log n)$ in $O(\log n)$ time [11]. Thus, we can apply binary search on the vertices (or the edges) on $\pi(s,t)$ efficiently.

Imagine that we rotate a line-of-sight along the vertices of $\pi(s,t)$ for two query points s and t in P. Lemma 1 implies that there is a line-of-sight containing s^* and t^*, where (s^*,t^*) is an optimal solution. We call it an *optimal line-of-sight*. We define the order of any two lines-of-sight as the order in which they appear during this rotational sweep process. By the following lemma, we can apply binary search on the sequence of events along ∂P and find two consecutive events such that the respective local optimum achieved between them is a global optimal solution.

▶ **Lemma 10.** *The geodesic distance between s (and t) and the rotating line-of-sight increases (and decreases) monotonically as the line-of-sight rotates along the vertices of $\pi(s,t)$ from s.*

Proof. Let ℓ be a line-of-sight which is tangent to $\pi(s,t)$ at a vertex v. Consider the subdivision of P induced by ℓ and let P_s be the subpolygon that contains s. Let ℓ' be a line-of-sight that comes after ℓ during the rotational sweep process. We claim that ℓ' does not intersect the interior of P_s. If ℓ' is tangent to $\pi(s,t)$ at v, it never intersects the interior of P_s as shown in the proof of Lemma 3. Assume that ℓ' is tangent to $\pi(s,t)$ at a vertex u that comes after v along $\pi(s,t)$ from s, but intersects the interior of P_s. Without loss of generality, assume that ℓ is horizontal and P_s lies locally below ℓ. Then u must lie strictly above the line containing ℓ. However, since both v and u are vertices of $\pi(s,t)$ and ℓ is tangent to $\pi(s,t)$ at v, there must be another vertex u' of $\pi(s,t)$ that lies on or below the line containing ℓ and appears between v and u along $\pi(s,t)$. Thus, u is not visible from any point on ℓ, and ℓ' does not intersect the interior of P_s. Since $\pi(s,\ell')$ intersects ℓ, we have $\pi(s,\ell') \geq \pi(s,\ell)$. The claim for t and the rotating line-of-sight can be shown analogously. ◀

5.1 Binary Search for the Path-Events

We first consider the path-events, and find two consecutive path-events containing an optimal line-of-sight between them. Let $\pi(s,t) := (s = v_0), v_1, \ldots, v_{k-1}, (t = v_k)$. Due to the shortest-path data structure by Guibas and Hershberger, we can obtain $\pi(s,t)$ represented as a binary tree of height $O(\log n)$ in $O(\log n)$ time. Consider an edge $\overline{v_i v_{i+1}}$ of $\pi(s,t)$. We can determine whether or not an optimal line-of-sight is tangent to $\pi(s,t)$ at a vertex lying after v_i along $\pi(s,t)$ in $O(\log n)$ time. To do this, we compute the line-of-sight ℓ containing $\overline{v_i v_{i+1}}$ in $O(\log n)$ time [12] and compute the length of $\pi(s,\ell)$ and $\pi(t,\ell)$ in $O(\log n)$ time [9]. An optimal line-of-sight is tangent to $\pi(s,t)$ at a vertex lying after v_i if and only if $\pi(s,\ell)$ is shorter than $\pi(t,\ell)$. Therefore, we can compute the two consecutive path-events with an optimal solution lying between them in $O(\log^2 n)$ time.

5.2 Binary Search for the Boundary-Events

Now we have the vertex v_i of $\pi(s,t)$ contained in an optimal line-of-sight. We find two consecutive boundary-events defined by lines-of-sight tangent to $\pi(s,t)$ at v_i such that an optimal line-of-sight lies between them. Let $\tilde{x}_i$ and x_i be the first points of ∂P hit by the

rays from any point in $\overline{v_{i-1}v_i}$ towards v_{i-1} and v_i, respectively. See Figure 4. Similarly, let $\tilde{x}_{i+1}$ and x_{i+1} be the first points of ∂P hit by the rays from any point in $\overline{v_iv_{i+1}}$ towards v_i and v_{i+1}, respectively. These four points of ∂P can be found in $O(\log n)$ time by the ray-shooting data structure [12]. Without loss of generality, we assume that a line-of-sight rotates around v_i in the counter-clockwise direction in the rotational sweep process. Let $\tilde{\gamma}$ be the part of ∂P lying from $\tilde{x}_i$ to $\tilde{x}_{i+1}$ in counter-clockwise order, and γ be the part of ∂P lying from x_i to x_{i+1} in counter-clockwise order. An optimal line-of-sight ℓ^* has one endpoint on $\tilde{\gamma}$ and the other endpoint on γ.

We first find the edge of $\tilde{\gamma}$ (resp. γ) containing an endpoint of ℓ^* by applying binary search on the vertices of $\tilde{\gamma}$ (resp. γ). This gives two consecutive boundary-events such that ℓ^* lies between them. We now show how to find the edge of γ containing an endpoint of ℓ^*. The edge on $\tilde{\gamma}$ can be found analogously.

We perform a binary search on the vertices in γ as follows. Let x^* be the endpoint of ℓ^* contained in γ. For any vertex u of γ, we can determine which part of γ with respect to u contains x^* in $O(\log n)$ time. To do this, we consider the line-of-sight ℓ containing the edge of $\pi(v_i, u)$ incident to v_i. Observe that ℓ intersects $\pi(v_i, u)$ only in the edge including its endpoints as $\pi(v_i, u)$ is a shortest path. See Figure 6(a). Since we can obtain the edge of $\pi(v_i, u)$ incident to v_i in $O(\log n)$ time using the shortest-path data structure, we can obtain ℓ in the same time. Here, to obtain the endpoint of ℓ on γ, we use the ray-shooting data structure that supports $O(\log n)$ query time [12]. Then we compare $d(s, \ell)$ and $d(t, \ell)$ in $O(\log n)$ time. The point x^* comes after u from x_i if and only if $d(s, \ell) < d(t, \ell)$. Therefore, we can determine which part of γ with respect to u contains x^* in $O(\log n)$ time, and thus the binary search is completed in $O(\log^2 n)$ time. In this way, we can compute two consecutive boundary-events such that an optimal line-of-sight lies between them in $O(\log^2 n)$ time.

5.3 Binary Search for the Bend-Events

Now we have two consecutive events in the sequence of all path- and boundary-events that contain an optimal line-of-sight ℓ^* between them. Let ℓ_1 and ℓ_2 be two lines-of-sight corresponding to the two consecutive events such that ℓ_2 comes after ℓ_1. The remaining task is to handle the bend-events lying between them. For the bend-events, we perform a binary search on the edges of $\pi(s, \ell_1) \cup \pi(s, \ell_2)$ in $O(\log^2 n)$ time. Then we perform binary search on the edges of $\pi(t, \ell_1) \cup \pi(t, \ell_2)$ in $O(\log^2 n)$ time. In the following, we describe the binary search on $\pi(s, \ell_1) \cup \pi(s, \ell_2)$. The other one can be done analogously.

We find the point s' such that $\pi(s, s')$ is the maximal common subpath of $\pi(s, \ell_1)$ and $\pi(s, \ell_2)$ from s in $O(\log n)$ time using the shortest-path data structure [11]. See Figure 6(b). Then we obtain $\pi' = \pi(s', \ell_1) \cup \pi(s', \ell_2)$ represented as a binary tree of height $O(\log n)$ in $O(\log n)$ time. For an edge e of π', we use $\ell(e)$ to denote the line-of-sight containing v_i and orthogonal to the line containing e. Observe that $\ell(e)$ comes after $\ell(e')$ if and only if e comes after e' along π' from ℓ_1. Also, given an edge e of π', we can compute $\ell(e)$ in constant time. Using these properties, we can find two consecutive edges e and e' of π' such that ℓ^* lies between $\ell(e)$ and $\ell(e')$ in $O(\log^2 n)$ time by applying binary search on π' as we did for path- and boundary-events.

Now we have two consecutive events in the sequence of all path-, boundary- and bend-events that contains ℓ^* between them. Recall that the combinatorial structure of $\pi(s, \ell)$ (and $\pi(t, \ell)$) is the same for every lines-of-sight lying between the two events. Let (u_s, w_s) and (u_t, w_t) be the edges of $\pi(s, \ell)$ and $\pi(t, \ell)$ incident to ℓ at w_s and w_t, respectively, for any line-of-sight ℓ lying between the two events. Using the shortest-path data structure, we can obtain $u_s, u_t, d(s, u_s)$ and $d(t, u_t)$ in $O(\log n)$ time. Then we apply the algorithm in

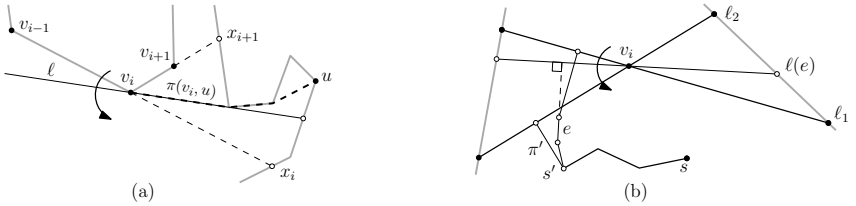


Figure 6 (a) The line-of-sight intersecting $\pi(v_i, u)$ contains the edge of $\pi(v_i, u)$ incident to v_i. (b) The maximal common subpath of $\pi(s, \ell_1)$ and $\pi(s, \ell_2)$ from s is $\pi(s, s')$.

Lemma 7 to find an optimal line-of-sight in constant time. In this way, we can obtain an optimal line-of-sight in $O(\log^2 n)$ time in total.

Therefore, we can find two consecutive events with an optimal solution between them, and we can obtain an optimal solution in $O(\log^2 n)$ time in total.

▶ **Theorem 11.** *Given a simple n-gon P, we can preprocess it in $O(n)$ time to find the minimum of the longer distance that s and t travel in order to see each other in P can be computed in $O(\log^2 n)$ time for any two query points $s, t \in P$.*

References

1. Esther M. Arkin, Alon Efrat, Christian Knauer, Joseph S. B. Mitchell, Valentin Polishchuk, Günter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016. doi:10.20382/jocg.v7i2a5.
2. Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *Discrete & Computational Geometry*, 27(4):461–483, 2002. doi:10.1007/s00454-001-0089-9.
3. Svante Carlsson, Håkan Jonsson, and Bengt J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22(3):377–402, 1999. doi:10.1007/PL00009467.
4. Wei-pang Chin and Simeon C. Ntafos. Optimum watchman routes. In Alok Aggarwal, editor, *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 24–33. ACM, 1986. doi:10.1145/10515.10518.
5. Wei-pang Chin and Simeon C. Ntafos. Optimum watchman routes. *Inf. Process. Lett.*, 28(1):39–44, 1988. doi:10.1016/0020-0190(88)90141-X.
6. Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 473–482. ACM, 2003. doi:10.1145/780542.780612.
7. Adrian Dumitrescu and Csaba D. Tóth. Watchman tours for polygons with holes. *Comput. Geom.*, 45(7):326–333, 2012. doi:10.1016/j.comgeo.2012.02.001.
8. Anurag Ganguli, Jorge Cortes, and Francesco Bullo. Visibility-based multi-agent deployment in orthogonal environments. In *Proceedings of the 2007 American Control Conference (ACC '07)*, pages 3426–3431, 2007. doi:10.1109/ACC.2007.4283034.
9. Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.
10. Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. doi:10.1007/BF01840360.

11 John Hershberger. A new data structure for shortest path queries in a simple polygon. *Inf. Process. Lett.*, 38(5):231–235, 1991. doi:10.1016/0020-0190(91)90064-O.

12 John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.

13 Ramtin Khosravi and Mohammad Ghodsi. The fastest way to view a query point in simple polygons. In *Proceedings of the 21st European Workshop on Computational Geometry*, pages 187–190, 2005.

14 Joseph S. B. Mitchell. Approximating watchman routes. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 844–855. SIAM, 2013. doi:10.1137/1.9781611973105.60.

15 Haitao Wang. Quickest visibility queries in polygonal domains. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, volume 77 of *LIPIcs*, pages 61:1–61:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.61.

16 Erik L. Wynters and Joseph S. B. Mitchell. Shortest paths for a two-robot rendez-vous. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 216–221, 1993.

Multistage Matchings

Evripidis Bampis
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France
evripidis.bampis@lip6.fr

Bruno Escoffier
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France
bruno.escoffier@lip6.fr

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, Paris, France
michail.lampis@dauphine.fr

Vangelis Th. Paschos
Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, Paris, France
vangelis.paschos@dauphine.fr

Abstract

We consider a *multistage* version of the Perfect Matching problem which models the scenario where the costs of edges change over time and we seek to obtain a solution that achieves low total cost, while minimizing the number of changes from one instance to the next. Formally, we are given a sequence of edge-weighted graphs on the same set of vertices V, and are asked to produce a perfect matching in each instance so that the total edge cost plus the transition cost (the cost of exchanging edges), is minimized. This model was introduced by Gupta et al. (ICALP 2014), who posed as an open problem its approximability for bipartite instances. We completely resolve this question by showing that *Minimum Multistage Perfect Matching* (Min-MPM) does not admit an $n^{1-\epsilon}$-approximation, even on bipartite instances with only two time steps.

Motivated by this negative result, we go on to consider two variations of the problem. In *Metric Minimum Multistage Perfect Matching* problem (Metric-Min-MPM) we are promised that edge weights in each time step satisfy the triangle inequality. We show that this problem admits a 3-approximation when the number of time steps is 2 or 3. On the other hand, we show that even the metric case is APX-hard already for 2 time steps. We then consider the complementary maximization version of the problem, *Maximum Multistage Perfect Matching* problem (Max-MPM), where we seek to maximize the total profit of all selected edges plus the total number of non-exchanged edges. We show that Max-MPM is also APX-hard, but admits a constant factor approximation algorithm for any number of time steps.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Perfect Matching, Temporal Optimization, Multistage Optimization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.7

Acknowledgements This work benefited from the support of the FMJH program "Gaspard Monge in optimization and Operation Research" and from the support to this program from EDF, via the project 2016-1760H/C16/1507 "Stability versus Optimality in Dynamic Environment Algorithmics".

1 Introduction

In classical Combinatorial Optimization, given an instance of a problem the goal is to find a solution optimizing the value of the objective function. However, in many applications the

instance may change over time and the goal is to find a tradeoff between the quality of the solution in each time step and the stability of the solution in consecutive time steps. As an example, consider an instance of an assignment problem, where the goal is to compute the best assignment of tasks to workers, assuming that we know the cost c_{ij} of performing task j by worker i. In the classical setting, it is possible to choose the assignment that minimizes the total cost in polynomial time. When the costs change over time (as for instance when a worker is not able to do some long task on a very busy day (infinite cost)) the optimal solutions of each time step may differ, inducing a transition cost for setting new task-worker pairs between two consecutive solutions. Hence, the naïve approach of finding a new optimal solution in each time step has the drawback that it does not take care of the penalty (transition cost) that is induced by the changes in the solution.

In this paper we study a multistage version of the *Perfect Matching problem* that follows this motivation and was originally introduced by Gupta, Talwar, and Wieder [11]. In this problem we are given a time horizon: $t = 1, 2, \ldots, T$ where for each time t we are given an instance G_t of *Perfect Matching* (that is, an edge-weighted graph) on the same set of vertices V. The goal is to determine a sequence of solutions $S = (M_1, M_2, \ldots, M_T)$ that both (1) are near-optimal (*quality*), and (2) induce small transition costs (*stability*). In other words, the goal is to determine a sequence of perfect matchings, one for each stage (time step) t, such that their total cost is small and the solution does not change too radically from one step to the next.

It was shown in [11] that this multistage problem is significantly harder than classical *Perfect Matching*. In fact, it is NP-hard to even approximate the optimal solution within $n^{1-\epsilon}$, for instances with only 8 times steps. Gupta et al. then posed as an explicit question whether the problem becomes easier for bipartite instances. Their work suggests also the question whether this hardness also applies for fewer than 8 steps. The bipartite restriction is especially interesting because Gupta et al. showed that related matroid-based optimization problems remain tractable for $T = 2$, and bipartite *Perfect Matching* can be seen as a matroid intersection problem. One could therefore hope that the matroid structure might make the bipartite case tractable for some small values of T, or at least approximable.

Our main contribution in this paper is to settle this question from [11] in the negative: we show that *Minimum Multistage Perfect Matching* (Min-MPM) is $n^{1-\epsilon}$-inapproximable, even for $T = 2$ time steps, unless $P = NP$. Motivated by this very negative result, we then investigate two other version of the problem: the *Metric Minimum Multistage Perfect Matching* problem (Metric-Min-MPM), where the input is guaranteed to satisfy the triangle inequality, and the *Maximum Multistage Perfect Matching* problem (Max-MPM), where we consider the complementary optimization objective.

Problem definition. Formally, the Min-MPM problem is defined as follows: We are given a sequence $G_1, \ldots, G_T$ of T undirected graphs, on the same set of vertices V. At each time step $1 \leq t \leq T$, the graph G_t is given with a cost function c_t on edges: $c_t(e) \in \mathbb{Q}_{\geq 0} \cup \{+\infty\}$. We are also given a transition cost $M \geq 0$. A solution is a sequence $S = (M_1, \ldots, M_t)$ where M_t is a perfect matching of G_t. Each solution (sequence) has two costs: a *matching cost* $c(S)$ and a *transition cost* $D(S)$. The goal is to minimize $c(S) + D(S)$. A matching M_t has a matching cost $c_t(M_t)$ which is equal to the sum of the costs of the edges of the perfect matching. The *matching cost* of S is $c(S) = \sum_{t=1}^{T} c_t(M_t)$. The *transition cost* is defined as $D(S) = \sum_{t=1}^{T-1} D_t$, where $D_t = M \cdot |M_{t+1} \setminus M_t|$ is proportional to the number of edges removed between time t and $t + 1$ – which is equal to the number of added edges since the matchings are perfect. Notice that by allowing infinite cost on edges we may assume w.l.o.g. the graphs to be complete.

In the Metric-Min-MPM, at each stage c_t obeys the triangle inequality: $c_t(u,v) + c_t(v,w) \geq c_t(u,w)$. Finally, in the Max-MPM version, we consider that $c_t(e)$ is the *profit* obtained by taking edge e (at time t). Then a solution sequence S has a *matching profit* $c(S) = \sum_t c_t(M_t)$. We define the *transition profit* $D(S)$ as $D(S) = \sum_{t \leq T-1} D_t$ where $D_t = M \cdot |M_{t+1} \cap M_t|$ is proportional to the number of edges that *remain* between time t and $t+1$. The goal now is to maximize $c(S) + D(S)$. Notice that in Max-MPM, we may no longer assume that the graphs are complete, since this assumption modifies the problem (we get profit by maintaining an edge, even of profit 0, from one time step to the next one).

Related work. A model that is close to our setting is the reoptimization model of Schieber et al. [15]. In their work, they are given a starting solution and a new instance and the goal is to minimize the sum of the cost of the new instance and of the transition cost. The model of multistage optimization that we use in this work has been studied earlier by Buchbinder et al. [5] and Buchbinder, Chen and Naor [4] for solving a set of fractional problems. Eisenstat et al. [7] studied a similar multistage optimization model for facility location problems. Their main result was a logarithmic approximation algorithm, which was later improved to a constant factor approximation by An et al. [1]. More broadly, many classical optimization problems have been considered in online or semi-online settings, where the input changes over time and the algorithm tries to adjust the solution (re-optimize) by making as few changes as possible. We refer the reader to [2, 3, 6, 10, 13, 14] and the references therein.

As mentioned, Gupta et al. [11] studied the Multistage Maintenance Matroid problem for both the offline and the online settings. Their main result was a logarithmic approximation algorithm for this problem, which includes as a special case a natural multistage version of Spanning Tree. The same paper also introduced the study of Min-MPM, which is the main problem we study here. They showed that the problem becomes hard to approximate even for a constant number of stages. More precisely, they showed the following result (n denotes the number of vertices in the graphs).

▶ **Theorem 1** ([11]). *For any $\epsilon > 0$,* Min-MPM *is not $n^{1-\epsilon}$-approximable unless $P = NP$. This holds even when the costs are in $\{0, \infty\}$, $M = 1$, and the number of time steps is a constant.*

Theorem 1 is proved for $T = 8$, starting from the fact that 3-colorability is NP-hard in graphs of maximum degree 4 [8]. The authors leave as an open question the approximability of the problem in bipartite graphs, and ask for subcases with better approximatibility behavior.

Our contribution. We answer the open question of [11] by showing that the problem is hard to approximate even for bipartite graphs and for the case of two steps ($T = 2$). Then, we focus on the case where the edge costs are metric within every time step (Metric-Min-MPM). On the negative side, we prove that the problem remains APX-hard even if $T = 2$. On the positive side, we show that Metric-Min-MPM admits a 3-approximation algorithm for two and three stages. Finally, for the maximization version of the problem, Max-MPM, we prove that it admits a constant factor approximation algorithm but is APX-hard.

2 Min-MPM for bipartite graphs

We answer the open question of [11] about the approximability of bipartite Min-MPM.

▶ **Theorem 2.** *For any $\epsilon > 0$,* Min-MPM *cannot be approximated within a factor of $n^{1-\epsilon}$, even if the input has $T = 2$ time steps, the input graphs are bipartite, $M = 1$ and the costs of edges are in $\{0, \infty\}$, unless P=NP.*

Using infinite costs, the same result immediately holds for bipartite complete graphs, as well as for complete graphs.

Proof. We give a gap-introducing reduction from Perfect 3DM (3-Dimensional Matching), known to be NP-complete [9]. We are given an instance of Perfect 3DM which consists of three sets X, Y, Z, with $|X| = |Y| = |Z| = n$, and a set Q of elements of $X \times Y \times Z$, with $|Q| = m \leq n^3$. We are whether there exists a subset of n pair-wise disjoint elements of Q, or not.

We construct an instance of our problem as follows: first, we create four sets of vertices A, B, C, D with $|A| = |B| = n$ and $|C| = |D| = m$. To ease notation suppose that the elements of our sets X, Y, Z, Q, A, B, C, D are labeled as $\{x_1, \ldots, x_n\}$, $\{y_1, \ldots, y_n\}$, $\{z_1, \ldots, z_n\}$, $\{q_1, \ldots, q_m\}$, $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_n\}$, $\{c_1, \ldots, c_m\}$, and $\{d_1, \ldots, d_m\}$ respectively.

For any $j \in \{1, \ldots, m\}$ we construct a set of $2n^{\lceil \frac{4}{\epsilon} \rceil}$ new vertices. We connect c_j to d_j through a path traversing all these vertices (thus this is a path from c_j to d_j with $2n^{\lceil \frac{4}{\epsilon} \rceil} + 2$ vertices). We set the cost of all the internal edges of these paths for both time-steps to 0.

For all $i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$ we do the following: if $x_i \in q_j$ we set the cost of the edge (a_i, c_j) to 0 in time step 1; if $y_i \in q_j$ we set the cost of the edge (a_i, c_j) to 0 in time step 2; if $z_i \in q_j$ we set the cost of the edge (b_i, d_j) to 0 in both time steps. All other edge costs are set to ∞ (or some other sufficiently large value). This completes the construction. Observe that the new graph has $5n + 3m + 2mn^{\lceil \frac{4}{\epsilon} \rceil}$ vertices, so at most $C \cdot n^{\frac{4}{\epsilon}+4}$ (for some constant C) since $m \leq n^3$. Note also that the new graph is bipartite because the paths that we added from c_j to d_j have odd lengths, hence the bipartition $(A \cup D, B \cup C)$ can be extended to a bipartition of the whole graph.

Suppose that the original instance has a set $Q' \subseteq Q$ such that $|Q'| = n$ and no element of $X \cup Y \cup Z$ appears in two elements of Q'. We obtain a multistage matching as follows: For each $q_j \in Q'$ such that $q_e = (x_{i_1}, y_{i_2}, z_{i_3})$ we use the edge (a_{i_1}, c_j) in step 1, the edge (a_{i_2}, c_j) in step 2, and the edge (b_{i_3}, d_j) in both time steps. Note that this fully specifies how the vertices of $A \cup B$ are matched. We now complete the matching by selecting a set of edges from the paths connecting each c_j to d_j: if $q_j \in Q'$, then both c_j, d_j have been matched to $A \cup B$ in both time steps, and we select in both time steps the unique perfect matching of the path connecting them; if $q_j \notin Q'$, then neither c_j, d_j is matched to $A \cup B$ in either time step, so we select the perfect matching on the path from c_j to d_j, including these two vertices. Observe that the cost of all edges we use is 0, while we only change at most n edges from one time step to the other, hence the total transition cost is at most nM.

Suppose that the original instance does not have a solution and consider any multistage matching in the new instance. We will show that it must make at least $n^{\frac{4}{\epsilon}}$ changes from one time step to the other. We will say that $q_j \in Q$ is selected in time step 1, if in that time step c_j is matched to an element of A. If q_j is selected in time step 1, then d_j is matched to an element of B in that time step, otherwise it would be impossible to have a perfect matching on the path connecting c_j to d_j. If some q_j is selected in time step 1, but not in time step 2, then the solution must change all internal edges on the perfect matching on the path from c_j to d_j, hence it makes at least $n^{\frac{4}{\epsilon}}$ changes, and we are done. What remains therefore to show is that if the solution maintains the set of selected q_j in the two time steps, then we can construct a solution to the original instance. Indeed, since all of $A \cup B$ is matched, we have n selected q_j's. Each element of $C \cup D$ has at most one edge connecting it to $A \cup B$ in each step, hence if it is selected this edge must be used. But if we select q_{j_1}, q_{j_2} that overlap, then two selected elements will have a common neighbor in $A \cup B$ and will therefore not be matched, contradiction.

Since the new graph has N vertices with $n^{\frac{1}{\epsilon}} \leq N \leq Cn^{\frac{4}{\epsilon}+4}$ vertices, it is NP-hard to distinguish if the optimal is at most $nM \leq N^{\epsilon}M$ or at least $n^{\frac{4}{\epsilon}}M \geq N^{1-\epsilon}M/C$. ◀

3 Metric-Min-MPM

We consider in this section that c_t obeys the triangle inequality: $c_t(u,v)+c_t(v,w) \geq c_t(u,w)$. In particular, the graph is complete. As seen before, the problem is hard to approximate even if there are only 2 time steps with general costs. We show here that while the problem is APX-hard in the metric case even with only 2 time steps (Section 3.1), it admits a 3-approximation algorithm in this case (2 time steps), see Section 3.2. We then extend this last result to the case of 3 time steps in Section 3.3.

3.1 APX-hardness for 2 time steps

In the case of 2 time steps the following result is proved.

▶ **Theorem 3.** Metric-Min-MPM *is APX-hard, even if the input has $T = 2$ time steps.*

Proof. We give a gap-preserving reduction from Max 3DM. We are given an instance of Max 3DM which consists of three sets X, Y, Z, with $|X| = |Y| = |Z| = n$, a set Q of elements of $X \times Y \times Z$, with $|Q| = m$, and an integer k. We are asked if there exists a subset of k pair-wise disjoint elements of Q. We assume that n, m and k are even (if not simply make two independent copies of the initial instance). This problem is APX-hard even if the occurence of each element is bounded above by a constant $C = 3$ [12]. Note that in this case the optimum value is at least $m/7$ (greedy algorithm; at most 6 incompatible triplets are removed when a triplet is chosen). So m, n and k are linearly related ($3n \geq m \geq k \geq m/7 \geq n/21$).

We construct an instance of Metric-Min-MPM as follows: first, we create five sets of vertices X, Y, Z, G, D with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, $Z = \{z_1, \ldots, z_n\}$, $G = \{g_1, \ldots, g_m\}$ and $D = \{d_1, \ldots, d_m\}$.

The graph is complete, and we set the following costs:

- At time step 1, Z is seen as a single point very far from the rest of the graph: (z_i, z_j) has cost 0 for $z_i, z_j \in Z$, and (z_i, v) has infinite cost for $z_i \in Z$, $u \notin Z$.
- The same is done for X at time 2.
- The m edges (g_i, d_i) have cost 1 at both time steps.
- For each triplet $q_i = (x_j, y_p, z_s)$: at time 1 edges (x_j, g_i) and (d_i, y_p) have cost a (a is a sufficiently large constant, to be specified later), and, for the triangle inequality to hold, (x_j, d_i) and (g_i, y_p) have cost $a + 1$. Similarly at time 2: (z_s, g_i) and (d_i, y_p) have cost a, and, for the triangle inequality to hold, (z_s, d_i) and (g_i, y_p) have cost $a + 1$.

All non yet defined costs are equal to $2a$. The transition cost is $M = 1$. Figure 1 gives an illustration of the construction.

Note that the triangle inequality holds in both time steps.

We show that (1) if there is a 3DM of size k then there exists a solution of Metric-Min-MPM whose total cost is at most $2m + 4an - k/2$, and (2) conversely from a solution of the multistage problem of total cost z we can construct a 3DM of size at least $2(2m + 4an - z)$. This proves APX-hardness since a is a constant, and m, n and k are linearly related.

Let us first prove (1), and suppose that we have a 3DM of size k, say (for ease of notation) $q_1, \ldots, q_k$ where $q_i = (x_i, y_i, z_i)$. Then we define a solution S of the multistage matching as follows:

- We take the $(m - k)$ edges of triplets (g_j, d_j) *not* in the 3DM, at both time steps 1 and 2;
- For $q_i, 1 \leq i \leq k$: we take edges (x_i, g_i) at time 1, (z_i, g_i) at time 2, and (y_i, d_i) at time 1 and 2.
- We match together the $(n - k)$ remaining vertices of Y, choosing the same $\frac{n-k}{2}$ edges at both time steps.

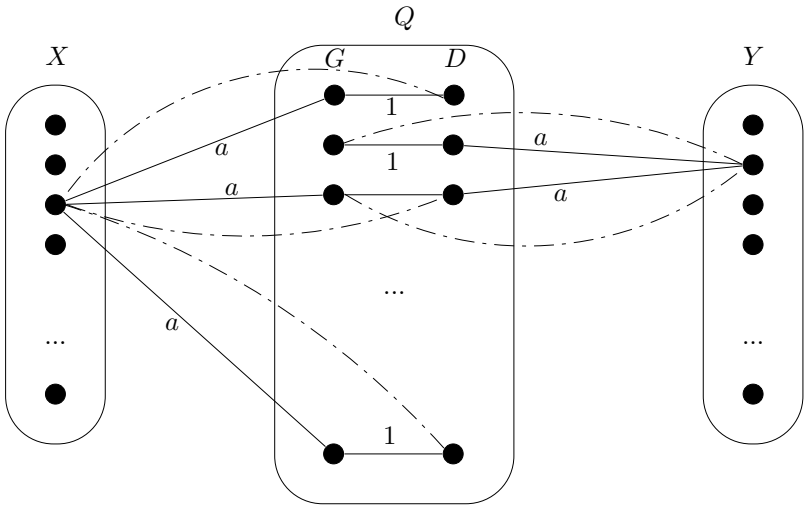


Figure 1 An illustration of the reduction at time $t = 1$, without representing Z - the construction is symmetric for time step $t = 2$. The third element of X is in the first, third and last triplet of Q. The second element of Y is in the second and third triplet. The dashed edges have costs $a + 1$. Not represented edges have cost $2a$..

- We match together the $(n - k)$ remaining vertices of X at time 1. At time 2 we keep these $\frac{n-k}{2}$ edges and match the remaining k vertices of X together.
- We do the same for Z.

We get a solution (M_1, M_2) whose costs are:

- At time 1, the matching cost is $(m - k) + 2ak + 2a\frac{n-k}{2} + 2a\frac{n-k}{2} = m + 2an - k$;
- The matching cost at time 2 is the same.
- The number of modifications is $3k/2$: k edges (x_i, g_i) become (z_i, g_i), and $k/2$ edges in Z disappear at time 1 ($k/2$ edges appear in X at time 2).

In all, (M_1, M_2) has cost $2m + 4an - k/2$.

Conversely, suppose that we have a solution (M_1^0, M_2^0) of total cost z for the instance of Metric-Min-MPM. We first structure this solution using local modifications, and then show how to derive a matching from it.

- *Replacement 1.* First, suppose that M_1^0 takes (at time 1) an edge (x_j, g_i) of cost $2a$ - so x_j is not in the i-th triplet q_i of Q. Then d_i is matched with a vertex v with an edge of cost at least a. By replacing (at time 1) (x_j, g_i) and (d_i, v) by (x_j, v) and (g_i, d_i) we get a matching cost for these two edges at most $2a + 1$ instead of (at least) $3a$. Even considering that the transition cost may have increased by two, this replacement does not increase the cost of the solution for $a \geq 3$. The same argument applies for an edge (x_j, d_i) (time step 1), an edge (y_j, d_i) or (y_j, g_i) (time step 1 or 2) and for an edge (z_j, g_i) or (z_j, d_i) in M_2^0.
- *Replacement 2.* Now, suppose that M_1^0 takes an edge of cost $2a$ in $G \cup D$, say (g_i, g_j) with $i \neq j$ (the very same argument works for the 2 other cases (g_i, d_j) and (d_i, d_j)). Let v and w be the neighbors of d_i and d_j in M_1^0. By replacing the three edges (g_i, g_j), (d_i, v) and (d_j, w) by (g_i, d_i), (g_j, d_j) and (v, w), we get a matching cost at most $(2a + 2)$ instead of (at least) $4a$. Even considering that the transition cost may have increased by three, this replacement does not increase the cost of the solution for $a \geq 5/2$. The same holds for M_2^0.

- *Replacement 3.* Last, suppose that edges (y_j, g_i) and (y_s, d_i) are both taken at time 1 and 2. This costs $2(a + a + 1) = 4a + 2$. Then we can take instead edges (g_i, d_i) and (y_j, y_s) at both time steps, with the same cost $2 + 2(2a) = 4a + 2$.

In this way, we transform (M_1^0, M_2^0) into a solution (M_1, M_2) of cost at most z such that:

- No g_i (and no d_i) is matched using an edge of cost $2a$ (replacements 1 and 2).
- g_i and d_i cannot be both matched to the same vertices at time 1 and 2, unless they are matched together (replacement 3).

We now show how to find a 3DM from this solution (M_1, M_2). Let:

- N_x and N_z be respectively the number of edges in $X \times (G \cup D)$ at time 1 and in $Z \times (G \cup D)$ at time 2.
- N_y^1 and N_y^2 be respectively the number of edges in $Y \times (G \cup D)$ at time 1 and time 2, among which λ_1 (resp., λ_2) are of cost $a + 1$.
- N_y be the number of edges in $Y \times (G \cup D)$ that are taken at both times 1 and 2.

At time 1, besides these $N_x + N_y^1$ edges and the $n/2$ edges of cost 0 (vertices of Z), the other edges of (M_1, M_2) have cost either 1 (edges (g_i, d_i)) or $2a$. Since $N_x + N_y^1$ vertices in $G \cup D$ are already matched at time 1, there are at most $\frac{2m - N_x - N_y^1}{2}$ edges of cost 1 at time 1. Similarly, there are at most $\frac{2m - N_z - N_y^2}{2}$ edges of cost 1 at time 2.

Then, computing the matching cost of (M_1, M_2) we have

$$\begin{aligned} c(M_1, M_2) &\geq a\left(N_x + N_z + N_y^1 + N_y^2\right) + \lambda_1 + \lambda_2 + \frac{4m - N_x - N_z - N_y^1 - N_y^2}{2} \\ &\quad + 2a\left(\frac{n - N_x + n - N_y^1 + n - N_z + n - N_y^2}{2}\right) \\ &\geq 2m + 4na + \lambda_1 + \lambda_2 - \frac{N_x + N_z + N_y^1 + N_y^2}{2}. \end{aligned}$$

Now, note that at time 1 at least $N_x + N_y^1 - N_y + \frac{N_z}{2}$ edges disappear, so $D(M_1, M_2) \geq N_x + N_y^1 - N_y + \frac{N_z}{2}$. Similarly, at least $N_z + N_y^2 - N_y + \frac{N_x}{2}$ edges appear at time 2. So $D(M_1, M_2) \geq N_z + N_y^2 - N_y + \frac{N_x}{2}$. Then,

$$D(M_1, M_2) \geq \frac{N_x + N_z + N_y^1 + N_y^2}{2} - N_y + \frac{N_x + N_z}{4}.$$

This gives:

$$z \geq c(M_1, M_2) + D(M_1, M_2) \quad \geq \quad 2m + 4na + \lambda_1 + \lambda_2 - N_y + \frac{N_x + N_z}{4}.$$

Now, consider the set of indices i such that edge (y_j, d_i) is taken at both time steps, or edge (y_j, g_i) is taken at both time steps. Since, thanks to the preprocessing, for a given i this cannot concern both d_i or g_i, we know that there are exactly N_y such indices (edges). Since there are $\lambda_1 + \lambda_2$ edges of cost $a + 1$ between Y and $G \cup D$, among these N_y indices at least $N_y - (\lambda_1 + \lambda_2)$ are such that: (1) edge (d_i, y_j) is used at both time steps (2) an edge (x_s, g_i) of cost a is used at time 1 (since no edge of cost $2a$ is used for vertices in G) and (3) an edge (z_p, g_i) of cost a is used at time 2.

In other words these at least $N_y - (\lambda_1 + \lambda_2)$ indices correspond to triplets of a 3DM. So we have a 3DM of size (at least) $k = N_y - (\lambda_1 + \lambda_2)$. Then, $N_x \geq N_y - (\lambda_1 + \lambda_2) = k$ and similarly $N_z \geq k$, so $\frac{N_z + N_x}{4} \geq \frac{k}{2}$. All together, we get

$$z \geq 2m + 4an - k + \frac{k}{2} = 2m + 4an - \frac{k}{2}.$$ ◀

3.2 A 3-approximation algorithm for 2 time steps

We now devise an approximation algorithm. Informally, this algorithm first guesses the number k of edges that an optimal solution keeps between steps 1 and 2. Then it computes a set of k edges with low matching cost that it maintains between time 1 and 2. Finally, it completes this set of k edges into two perfect matchings, in such a way that, using the triangle inequality, the matching cost does not increase too much.

Formally, the algorithm `Metric2` runs the following procedure for k from 0 to $n/2$.

1. Let G_{1+2} be the graph where the edge costs are $c(u,v) = c_1(u,v) + c_2(u,v)$. Compute a minimum cost matching M^k of size exactly k in G_{1+2}.
2. Compute a minimum cost perfect matching M_1 in G_1, and a minimum cost perfect matching M_2 in G_2.
3. Consider the symmetric difference of the two matchings M^k and M_1 in G_1. This is a (vertex disjoint) set of paths $P_1, \dots, P_p$ and cycles. Define M_1^k as M^k plus the p edges linking the first vertex and last vertex of each path P_j.
4. Do the same to get M_2^k.
5. Consider $S^k = (M_1^k, M_2^k)$.

`Metric2` outputs the best solution S^k.

▶ **Theorem 4.** *`Metric2` is a (polytime) 3-approximation algorithm for* Metric-Min-MPM *when* $T = 2$.

Proof. We first prove that S^k is a feasible solution, i.e., M_i^k is a perfect matching of G_i. Since M_i is a perfect matching, in all paths P_j the first and last edges belong to M_i. Hence the first and last vertices are not covered by M^k, so M_1^k is a matching. Every other vertex is covered by M^k, so the matching is perfect.

Now, let us prove the claimed approximation ratio. Let us denote $S^* = (M_1^*, M_2^*)$ be an optimal solution, and consider S^k where $k = |M_1^* \cap M_2^*|$.

Since at least M^k is common between M_1^k and M_2^k, at least k edges are maintained between time 1 and 2 in S^k, as in S^*. So:

$$D(S^k) \leq D(S^*). \tag{1}$$

Now, let us prove that:

$$c_1(M_1^k) + c_2(M_2^k) \leq 3c_1(M_1^*) + 3c_2(M_2^*). \tag{2}$$

Thanks to the triangle inequality, in a path $P = (v_0, v_1, \dots, v_t)$, $c_i(v_0, v_t) \leq \sum_j c_i(v_j, v_{j+1})$: when adding edges (v_0, v_t) we add in total at most the total length of the paths, hence at most $c_i(M_i) + c_i(M^k)$. So $c_i(M_i^k) \leq c_i(M_i) + 2c_i(M^k)$. Using that $c_i(M_i) \leq c_i(M_i^*)$, we get:

$$c_1(M_1^k) + c_2(M_2^k) \leq c_1(M_1^*) + c_2(M_2^*) + 2(c_1(M^k) + c_2(M^k)).$$

By optimality of M^k and since S^* has k common edges between times 1 and 2, these k common edges induce a cost in S^* at least $c_1(M^k) + c_2(M^k)$. Then:

$$c_1(M_1^k) + c_2(M_2^k) \leq c_1(M_1^*) + c_2(M_2^*) + 2(c_1(M_1^*) + c_2(M_2^*))$$

and Equation 2 follows. From Equations 1 and 2 we derive:

$$c(S) + D(S) \leq 3c(S^*) + D(S^*).$$

The result immediately follows. ◀

3.3 A 3-approximation algorithm for 3 time steps

We now extend the previous result to the case of $T = 3$. As previously, if an optimal solution preserves in total k edges (operates in total $n - k$ modifications between time steps 1 and 2, and 2 and 3) we would like to first compute a set of k 'preserved' edges inducing a low cost, and then to complete this set as perfect matchings in each of the time steps. Now things get more complex since an edge can be preserved between steps 1 and 2, between steps 2 and 3, or during the whole process. It seems hard to mimic an optimal solution on these 3 types of edges (while inducing a low matching cost), but this difficulty can be overcome as follows.

Let G be the graph with edge cost $w = \min\{c_1 + c_2 + c_3, c_1 + c_2 + M, c_2 + c_3 + M\}$. If the minimum is $c_1 + c_2 + c_3$ (resp., $c_1 + c_2 + M$, $c_2 + c_3 + M$) we say that the edge is of type 1 (resp., 2, 3). Intuitively, edges of type 1 will be taken in steps 1, 2 and 3, edges of type 2 (resp., 3) will be taken in steps 1 and 2 (resp., 2 and 3). We present a 3-approximation algorithm `Metric3`. It runs the following procedure for k from 0 to $n/2$.

1. Compute a minimum cost matching M^k of size exactly k in G. Denote M_1^k the set of edges of M^k of type 1 or 2, $M_2^k = M^k$ and M_3^k the set of edges of M^k of type 1 or 3.
2. Compute a minimum cost perfect matching M_i in G_i, $i = 1, 2, 3$.
3. Consider the symmetric difference of the two matchings M_i^k and M_i in G_i. This is a (vertex disjoint) set of paths $P_1, \ldots, P_p$ and cycles. Define $M_i'^k$ as the set of p edges linking the first vertex and last vertex of each path P_j.
4. Consider $S^k = (M_1^k \cup M_1'^k, M_2^k \cup M_2'^k, M_3^k \cup M_3'^k)$.

Then `Metric3` outputs the best solution S^k.

▶ **Theorem 5.** *`Metric3` is a (polytime) 3-approximation algorithm for* Metric-Min-MPM *when* $T = 3$.

Proof. We first note that, as in the case for $T = 2$ time steps, $M_i^k \cup M_i'^k$ is a perfect matching of G_i, so S^k is a feasible solution.

Now let us deal with the approximation ratio. Let $S^* = (M_1^*, M_2^*, M_3^*)$ be an optimal solution. Let us consider the set $H = (M_1^* \cap M_2^*) \cup (M_2^* \cap M_3^*)$ of edges in S^* that are in (at least) two consecutive steps. Note that H is a matching (it is included in M_2^*). Consider S^k where $k = |H|$. We now prove the following result:

▶ **Lemma 6.** $D(S^k) + \sum_i c_i(M_i^k) \leq D(S^*) + c(S^*)$.

Proof. To prove this, let $k_1 = |M_1^* \cap M_2^* \cap M_3^*|$ be the number of edges in S^* that are taken at each of the 3 time steps. Hence, $k - k_1$ edges are taken at (only) 2 consecutive time steps. So there are $(n/2 + n/2 - 2k_1 - (k - k_1))$ modifications in total, and:

$$D(S^*) = M(n - k - k_1). \tag{3}$$

Recall that in G, $w = min\{c_1 + c_2 + c_3, c_1 + c_2 + M, c_2 + c_3 + M\}$. k_1 edges of H are present on the 3 time steps (matching cost $c_1 + c_2 + c_3$), while $k - k_1$ are present in two consecutive time steps (matching cost $c_1 + c_2$ or $c_2 + c_3$).

$$w(H) \leq c(S^*) + M(k - k_1). \tag{4}$$

Similarly, let λ_1 be the number of edges of type 1 in M^k. There are $(k - \lambda_1)$ edges of type 2 or 3, hence

$$w(M^k) = \sum_i c_i(M_i^k) + M(k - \lambda_1). \tag{5}$$

Indeed, in G cost c_1 applies to edges of type 1 and 2 ($c_1(M_1^k)$), cost c_2 applies to all edges of M^k ($c_2(M_2^k)$), cost c_3 applies to edges of type 1 and 3 ($c_3(M_3^k)$), and cost M to the $(k-\lambda_1)$ edges of type 2 and 3.

Also, the number of preserved edges in S^k is at least $k+\lambda_1$, so:

$$D(S^k) \leq M(n-k-\lambda_1). \tag{6}$$

Since H is a matching, in G we have $w(H) \geq w(M^k)$. This gives using Equations 4 and 5:

$$\sum_i c_i(M_i^k) + M(k-\lambda_1) \leq c(S^*) + M(k-k_1)$$

so $\sum_i c_i(M_i^k) \leq c(S^*) + M(\lambda_1 - k_1)$. Then using Equations 3 and 6 we get:

$$\sum_i c_i(M_i^k) + D(S^k) \leq c(S^*) + M(\lambda_1 - k_1) + M(n-k-\lambda_1) = c(S^*) + M(n-k-k_1)$$

$$= c(S^*) + D(S^*)$$

which concludes the proof of Lemma 6. ◀

Now, by triangle inequality, and the fact that $c_i(M_i) \leq c_i(M_i^*)$, we know that:

$$c_i(M_i'^k) \leq c_i(M_i^*) + c_i(M_i^k). \tag{7}$$

Then, from Lemma 6 and Equation 7 we get:

$$\begin{aligned} c(S^k) + D(S^k) &= \sum_i (c_i(M_i^k) + c_i(M_i'^k)) + D(S^k) \leq \sum_i (2c_i(M_i^k) + c_i(M_i^*)) + D(S^k) \\ &\leq c(S^*) + 2\left(\sum_i c_i(M_i^k) + D(S^k)\right) \leq 3c(S^*) + 2D(S^*). \end{aligned}$$

The result follows. ◀

4 Max-MPM

In the maximization version, we consider that $c_t(e)$ is the *profit* obtained by taking edge e (at time t). Then a solution sequence S has a matching profit $c(S) = \sum_t c_t(M_t)$. We define the transition profit $D(S)$ as $D(S) = \sum_{t \leq T-1} D_t$ where $D_t = M \cdot |M_{t+1} \cap M_t|$ is proportional to the number of edges that *remain* between time t and $t+1$. The goal now is to maximize $c(S) + D(S)$. Recall that in the maximization version we may no longer assume that the graphs are complete.

4.1 APX-hardness for 2 time steps

We first show that Max-MPM, even in the case of 2 time steps is APX-hard.

▶ **Theorem 7.** Max-MPM *is APX-hard even if* $T = 2$.

Proof. As previously, we consider the maximum 3DM problem in the case where the occurrence of each element is bounded by 3, hence the optimal value, the number of triplets and the size of the ground sets are linearly related.

Given three sets X, Y, Z each of size n, and m triplets q_i of $X \times Y \times Z$, we build two graphs G_1 and G_2 with $n' = 2m + 4n$ vertices:

- 4 sets D, E, F, G of size n;
- 2 sets $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_m\}$ of size m.

Vertices of D will represent elements of X, vertices of E and F elements of Y (twice), vertices of G elements of Z. Each triplet q_i is represented by one edge (a_i, b_i) in both graphs. It has cost 0.

If a triplet q_i is (x_j, y_k, z_l) then:

- In G_1 we put edges (d_j, a_i) and (b_i, e_k), both with cost M';
- In G_2 we put edges (f_k, a_i) and (b_i, z_l), both with cost M'.

Note that vertices in F, G have degree 0 in G_1, vertices in D, E have degree 0 in G_2.

We fix $M' = \frac{M+1}{4}$, and $M \geq 3$.

Let us show that there is a 3DM of size (at least) k if and only if there is a solution of profit at least $Mm + k$.

Suppose first that there is a set S of k independent triplets. Then we build matchings (M_1, M_2) as follows:

- if q_i is not in S, we take (a_i, b_i) both in M_1 and M_2. This gives transition profit $M(m-k)$.
- if $q_i = (x_j, y_k, z_l)$ is in S, then we take in M_1 the two edges (d_j, a_i) and (b_i, e_k), and in M_2 the two edges (f_k, a_i) and (b_i, z_l). This gives a matching profit $4kM'$.

Note that since any element of X, Y, Z is in at most one triplet of S, vertices in D, E, F, G are adjacent to at most one chosen edge. In other words M_1 and M_2 are matchings.

The profit of the solution is $4kM' + M(m-k) = k(M+1) + M(m-k) = Mm + k$.

Suppose now that there is a solution (M_1, M_2) of profit at least $Mm + k$. Suppose first that there is an edge (a_i, b_i) which is in M_1 but not in M_2. Then we get no transition profit for this edge. In M_2 we have taken at most one edge incident to a_i, and one edge incident to b_i, with matching profit at most $2M'$. Since these edges are not in G_1 they cannot give transition profit. So we can put in M_2 the edge (a_i, b_i) and remove the edges incident to a_i and b_i (if any). The profit increases by $M - 2M' = M/2 - 1/2 \geq 0$.

So we can assume that M_1 and M_2 have the same set of edges between A and B. Suppose now that there are two edges (a_i, b_i) and (a_s, b_s) both not in M_1 (equiv. not in M_2) corresponding to two intersecting triplets. Suppose for instance that x_j is in both triplets. This means that in M_1 we cannot take both edges (c_j, a_i) and (c_j, a_s), for instance (c_j, a_s) is not in M_1. Then we can add (a_s, b_s) is M_1 and M_2, and remove the (at most) 3 incident edges. This increases profit by $M - 3M' \geq 0$.

So, the set of edges (a_i, b_i) not in M_1 (or not in M_2) corresponds to a set of independent triplets. Let t the number of such edges. Since M_1 is a matching, besides these edges between A and B, there is at most two edges for each (a_i, b_i) not in M_1. Similarly, there is at most two edges in M_2 for each (a_i, b_i) not in M_2. So the matching profit is at most $4tM'$, and the transition profit is $M(m-t)$. The profit is $M(m-t) + 4tM' = Mm + t \geq Mm + k$. So $t \geq k$. ◀

4.2 Constant factor approximation algorithms

▶ **Theorem 8.** Max-MPM *is 1/2-approximable. If $T = 2$ it is 2/3-approximable, if $T = 3$ it is 3/5-approximable.*

Proof. Note that if the graphs are assumed to be complete (bipartite complete) then the ratio 1/2 is easily achievable. Indeed, consider two solutions:

- The first one S_1 consisting of the same perfect matching M_0 at all time steps;
- The second one S_2 consisting of a matching $\hat{M}_t$ of maximum profit on G_t for each t.

Output the best one.

Let $S^* = (M_1^*, \ldots, M_T^*)$ be an optimal solution. Clearly the profit of S_1 is at least the transition profit $D(S^*)$ of S^*. Also, $c(M_i^*) \leq c(\hat{M}_i)$ so the matching profit of S^* is at most the one of S_2. The ratio 1/2 follows.

If the graphs are not assumed to be complete things get harder since one cannot trivially optimize the transition profit by keeping a perfect matching along the multistage process.

Let us consider three consecutive time steps $t-1, t, t+1$. Let us consider the graph G'_t which is the same as G_t up to the profit on edges, which is now $c'_t(e)$ where:

1. $c'_t(e) = c_t(e) + 2M$ if e is in G_{t-1} and G_{t+1};
2. otherwise, $c'_t(e) = c_t(e) + M$ if e is in G_{t-1} or G_{t+1};
3. otherwise $c'_t(e) = c_t(e)$.

Let us consider a matching M'_t of maximum profit in G'_t.

▶ **Lemma 9.** $c'_t(M'_t) \geq D_{t-1}(S^*) + c_t(M_t^*) + D_t(S^*)$.

Proof. Let us consider the profit of M_t^* on G'_t. Since the set of edges preserved from time $t-1$ to time t is included in M_t^*, the profit $D_{t-1}(S^*)$ appears in the profit of M_t^* on G'_t ($+M$ on each common edges between the two consecutive graphs). This is also the case for $D_t(S^*)$, for the same reason. Of course, the profit $c_t(e)$ appears as well. Since M'_t is of maximum profit, the Lemma follows. ◀

Because of Lemma 9, choosing the matching M'_t at time steps $t-1$, t and $t+1$ in a solution generates a profit at least $D_{t-1}(S^*) + c_t(M_t^*) + D_t(S^*)$.

Note that, with similar arguments, if two times steps $t, t+1$ are involved, we can compute a matching H_i that we take at time steps $t, t+1$ generating a profit at least $c_t(M_t^*) + D_t(S^*)$. Symmetrically, we can compute a matching H'_i that we take at time steps $t, t+1$ generating a profit at least $c_{t+1}(M_t^*) + D_t(S^*)$.

Now we consider the following 2 solutions:

- S_1 consists of choosing H_1 at steps $1, 2$, H_3 at step $3, 4$, ... If T is even then we are done, otherwise we take an optimal matching $\hat{M}_T$ at step T.
- S_2 consisting of choosing an optimal matching $\hat{M}_1$ at step 1, then H_2 at steps $2, 3$, H_4 at steps $4, 5$,.... If T is even we take an optimal matching $\hat{M}_T$ at step T.

Output the best of these two solutions. Then: S_1 covers the transition profit of an optimal solution D_t for t odd, plus the matching profits for t odd. S_2 covers the transition profit of an optimal solution D_t for t even, plus the matching profits for t even. The ratio 1/2 follows.

Improvement for $T = 3$. The previous solutions S_1 and S_2 have profit (respectively) at least $c_1(S^*) + D_1(S^*) + c_3(S^*)$ and $c_1(S^*) + D_2(S^*) + c_2(S^*)$. S_3 takes $\hat{M}_1$ at step 1 and H'_2 at time steps 2 and 3, with profit at least $c_1(S^*) + D_2(S^*) + c_3(S^*)$; S_4 takes H'_1 at steps 1 and 2, and $\hat{M}_3$ at step 3, with profit at least $D_1(S^*) + c_2(S^*) + c_3(S^*)$. S_5 uses M'_2 at the 3 steps with profit at least $D_1(S^*) + c_2(S^*) + D_2(S^*)$ (thanks to Lemma 9). Take the best of these 5 solutions, and the ratio follows.

Improvement for $T = 2$. Simply take 3 solutions: S_1 is defined as previously, with profit at least $c_1(S^*) + D_1(S^*)$. S_2 takes H'_1 at both steps with profit at least $D_1(S^*) + c_2(S^*)$. S_3 consists of one optimal matching at step 1, and an optimal matching at step 2, with profit at least $c_1(S^*) + c_2(S^*)$. The ratio 2/3 follows. ◀

5 Concluding remarks

Following the results of Section 3, we leave as an open question the existence of a constant factor approximation algorithm for the metric case for a number of time steps bigger than 3. Also, we considered here an off-line version of the problem where the whole set of instances is known in advance. It would be worth investigating the on-line case where data are not known in advance.

References

1 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.
2 Barbara M. Anthony and Anupam Gupta. Infrastructure leasing problems. In *IPCO*, volume 4513 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 2007.
3 Nicolas K. Blanchard and Nicolas Schabanel. Dynamic sum-radii clustering. In *WALCOM*, volume 10167 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2017.
4 Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive analysis via regularization. In *SODA*, pages 436–444. SIAM, 2014.
5 Niv Buchbinder, Shahar Chen, Joseph Naor, and Ohad Shamir. Unified algorithms for online learning and competitive analysis. *Math. Oper. Res.*, 41(2):612–625, 2016.
6 Edith Cohen, Graham Cormode, Nick G. Duffield, and Carsten Lund. On the tradeoff between stability and fit. *ACM Trans. Algorithms*, 13(1):7:1–7:24, 2016.
7 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *ICALP (2)*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.
8 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete problems. In *STOC*, pages 47–63. ACM, 1974.
9 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
10 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016.
11 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer, 2014.
12 Viggo Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Inf. Process. Lett.*, 37(1):27–35, 1991.
13 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016.
14 Chandrashekhar Nagarajan and David P. Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013.
15 Baruch Schieber, Hadas Shachnai, Gal Tamir, and Tami Tamir. A theory and algorithms for combinatorial reoptimization. *Algorithmica*, 80(2):576–607, 2018.

Convex Hulls in Polygonal Domains

Luis Barba
Department of Computer Science, ETH Zürich, Zürich, Switzerland
luis.barba@inf.ethz.ch

Michael Hoffmann
Department of Computer Science, ETH Zürich, Zürich, Switzerland
hoffmann@inf.ethz.ch
https://orcid.org/0000-0001-5307-7106

Matias Korman[1]
Tohoku University, Sendai, Japan
mati@dais.is.tohoku.ac.jp

Alexander Pilz[2]
Department of Computer Science, ETH Zürich. Zürich, Switzerland
alexander.pilz@inf.ethz.ch
https://orcid.org/0000-0002-6059-1821

Abstract

We study generalizations of convex hulls to polygonal domains with holes. Convexity in Euclidean space is based on the notion of shortest paths, which are straight-line segments. In a polygonal domain, shortest paths are polygonal paths called *geodesics*. One possible generalization of convex hulls is based on the "rubber band" conception of the convex hull boundary as a shortest curve that encloses a given set of sites. However, it is NP-hard to compute such a curve in a general polygonal domain. Hence, we focus on a different, more direct generalization of convexity, where a set X is *geodesically convex* if it contains all geodesics between every pair of points $x, y \in X$. The corresponding *geodesic convex hull* presents a few surprises, and turns out to behave quite differently compared to the classic Euclidean setting or to the geodesic hull inside a simple polygon. We describe a class of geometric objects that suffice to represent geodesic convex hulls of sets of sites, and characterize which such domains are geodesically convex. Using such a representation we present an algorithm to construct the geodesic convex hull of a set of $O(n)$ sites in a polygonal domain with a total of n vertices and h holes in $O(n^3h^{3+\varepsilon})$ time, for any constant $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases geometric graph, polygonal domain, geodesic hull, shortest path

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.8

1 Introduction

Convexity is a fundamental concept in geometry and optimization, and computing the convex hull of a point set in the plane is a classic textbook problem in algorithm design. The convex hull of a set $S \subset \mathbb{R}^2$ is usually defined as the inclusion-minimal convex set that contains S, and showing that this statement is well-defined is a textbook exercise in itself. If S is finite,

[1] Supported in part by MEXT KAKENHI Nos. 17K12635, 15H02665, and 24106007.
[2] Supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35.

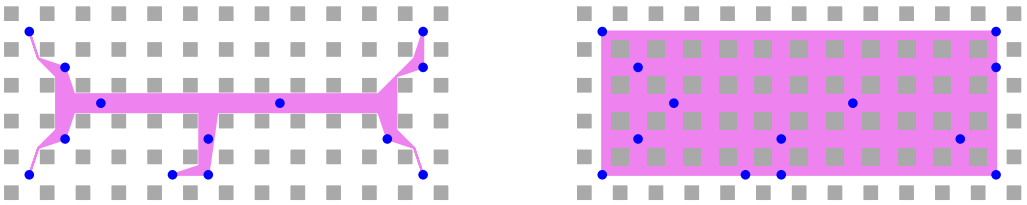


Figure 1 Two possible definitions of "convex hull" in a polygonal domain. The domain is shown in white, obstacles in gray, and sites are shown as blue dots. The left image depicts the relative hull, bounded by a curve of minimum length that separates the set of sites from the boundary. The right image depicts the definition we will use in this paper: an inclusion-minimal subset of the domain that contains all sites and all shortest paths between any two of its points.

the convex hull of S is a convex polygon. The boundary of this polygon describes the shortest path enclosing S, yielding an equivalent definition of the convex hull.

The definition of convexity builds on shortest paths: a set X is convex if for every pair $x, y \in X$ the shortest path between x and y is contained in X. Hence, convexity directly generalizes to any domain that has a notion of a shortest path. In the Euclidean setting, shortest paths are straight-line segments. But there is a variety of other domains that have a sensible notion of a shortest path. Specifically, shortest paths inside a simple polygon have been studied in the computational geometry literature.

A set R is called geodesically convex w.r.t. a polygon P if the shortest path in P between two points in R is also contained in R. Toussaint [20] studied how properties of point sets extend to geodesic environments. He introduced the geodesic convex hull of a set of points (called *sites*) inside a simple polygon P; it is the inclusion-minimal geodesically convex set containing the sites. Among several results, he showed how to compute the geodesic convex hull of k sites in a simple n-gon in $O((n + k) \log(n + k))$ time. Note that the geodesic convex hull properly generalizes the convex hull of a point set S; if we choose P to be, say, a bounding box of S, we obtain the convex hull of S.

A classic metaphor for the convex hull boundary is a "rubber band", describing the continuous transformation of a curve containing the sites to a homotopy-equivalent curve of minimal length. For geodesic convex hulls within a simple polygon P, the boundary ∂P is equivalent to the shortest cycle that separates the sites from ∂P [20]. However, if we consider sites in a polygonal domain with holes, this correspondence does not generalize.

We thus face (at least) two different ways to generalize the concept of a convex hull to general polygonal domains. On the one hand, we have the *(geodesic) convex hull* as an inclusion-minimal geodesically convex set that contains all sites (and may enclose holes). On the other hand, we have a shortest curve that separates the sites from the boundary, also called the *relative hull* of the sites. See Figure 1 for illustrations.

Both generalizations are interesting in their own right. The former definition is much more directly tied to the notions of convexity and shortest paths. Therefore this is how we propose to generalize the concept *convex hull* to general polygonal domains. The latter definition using relative hulls turns out rather unwieldy. For a set S of general sites inside a polygonal domain, a relative hull is not necessarily unique and NP-hard to compute. This follows from a slight modification of a result by Eades and Rappaport [12], who show that it is NP-hard to find the shortest curve separating two point sets. (A reduction from the rectilinear Steiner tree problem is also straightforward.)

Related work. Relative hulls have been studied in general polygonal domains, but only for a set of connected sites. Given a set of disjoint simple polygons with n vertices overall, de Berg [10] showed how to compute the shortest curve that separates one of these polygons from the others in $O(n \log n)$ time. Effectively, the algorithm computes the shortest cycle within a polygonal domain that separates a polygon P from the boundary. The proof directly generalizes to the case where P is an arbitrary outerplane graph. In a similar fashion, Mitchell et al. [18] compute the relative hull of paths in polynomial time.

In addition to Toussaint's generalization of the diameter, center, and median to the geodesic setting [20], separators [11], ham-sandwich cuts [7], spanning trees, Hamiltonian cycles and perfect matchings [6] have been generalized to point sites in simple polygons. Any concept defined on the order type of a point set allows for a generalization [2]. In general polygonal domains, the complexity of these problems increases substantially. Many problems become NP-hard, and where polynomial algorithms are known, the known bounds are nowhere near to what is known for simple polygons. For example, the diameter and center of a simple polygon can be computed in linear time [1, 14]. However, for a general domain, the best known algorithms use $O(n^{7.73})$ [3] and $O(n^{11} \log n)$ [4, 21] time, respectively.

Computing shortest paths in polygonal domains has been an active area of research (cf. [17]). While a single shortest path can be computed in $O(n \log n)$ time [15], data structures that support two-point shortest path queries in logarithmic time require a significant storage overhead. The state of the art data structure, allowing $O(\log n)$ query time, uses $O(n^{11})$ space and preprocessing time [9]. For points on the boundary of the domain, Bae and Okamoto [5] presented a data structure with logarithmic query time using $O(n^{5+\varepsilon})$ space and preprocessing time. A variant of their result is used as a subroutine in our algorithm.

Generalizations of convex hulls of point sets have also been considered in other settings. For example, Lubiw et al. [16] consider convex hulls in 2-dimensional globally non-positively curved polyhedral complexes. Such spaces have a unique shortest path between any two points. They pose as an open problem the study of convexity in domains where more than one shortest path between two points may exist. Our work is a step in this direction.

Results. We consider the inclusion-minimum geodesically convex set that contains a given set of sites in a polygonal domain. It is the first study of this natural generalization of convex hulls. Not even domains with a single hole have been considered so far (see also [16], where the problem is mentioned). It turns out that the problem of computing the geodesic convex hull within a polygonal domain is significantly more complex than within a simple polygon. Within a simple polygon, the structure of the geodesic convex hull only depends on the order type of the sites and the vertices, i.e., the orientation of point triples. In general polygonal domains, the homotopy of the shortest path between two sites depends on actual distances between sites and vertices. In particular, all direct attempts to discretize the problem failed. The examples given in Section 2 illustrate the differences to classic convexity and demonstrate how naive attempts to compute the geodesic convex hull fail.

As a main result, we characterize geodesically convex sets. To this end, we define a class of geometric objects, called *cactus domains*, and show that this class contains all geodesic convex hulls of finite sets of sites inside polygonal domains. More specifically, we use two concepts (called divisibility and tightness), and show that they are sufficient and necessary for a cactus domain to be geodesically convex. We provide algorithms to efficiently test both properties, resulting in a polynomial-time algorithm to compute geodesic convex hulls.

▶ **Theorem 1.** *Let P be a polygonal domain with n vertices and h holes, and let $S \subset P$ be a set of $O(n)$ sites. The geodesic convex hull of S in P can be computed in $O(n^3 h^{3+\varepsilon})$ time.*

While the running time of our algorithm might look high at first sight, it must be compared with algorithms and data structures that encode all geodesic paths in polygonal domains. In this direction, one must consider the state-of-the-art structure developed by Chiang and Mitchell [9] that uses $O(n^{11})$ space and preprocessing time; or the structure of Bae and Okamoto [5] using $O(n^{5+\varepsilon})$ space and preprocessing time for paths connecting points on the boundary. While no lower bounds are known, it is clear that the complexity of these problems is high and still far from being understood.

To improve upon the running time stated in Theorem 1, more structural insights would be required. As a first step in this direction, one could ask if a simpler algorithm can be designed to test whether a point lies in the geodesic convex hull of a set of sites in a polygonal domain.

2 Preliminaries

Polygonal domains. A *simple polygon* is a compact subset of $\mathbb{R}^2$ that is bounded by a simple closed curve formed by a finite number of line segments. For a simple polygon P denote by $\mathsf{V}(P)$ the set of its *vertices*, by $\mathsf{int}(P)$ the *interior* of P, and by ∂P its *boundary*. A *polygonal domain* P is defined by a finite collection $(P_0, P_1, \ldots, P_h)$ of $h+1$ simple polygons with the following properties: (1) $P_i \subset \mathsf{int}(P_0)$, for each $i > 0$, and (2) $P_i \cap P_j = \emptyset$, for all $i, j > 0$ with $i \neq j$. We say that P_0 and ∂P_0 are the *outer polygon* and *outer boundary* of P, respectively. The *boundary* of P is $\partial P = \bigcup_{i=0}^{h} \partial P_i$, the *interior* of P is $\mathsf{int}(P) = \mathsf{int}(P_0) \setminus \bigcup_{i=1}^{h} P_i$, the *vertices* of P are $\mathsf{V}(P) = \bigcup_{i=0}^{h} \mathsf{V}(P_i)$, and collectively $P = \mathsf{int}(P) \cup \partial P$. The polygons $P_1, \ldots, P_h$ are also referred to as *holes* of P. We also use the notation $P = (P_0, P_1, \ldots, P_h)$ to indicate that P is defined by the polygons $P_0, P_1, \ldots, P_h$ (although in principle we regard P as a subset of the plane rather than a tuple of polygons).

Geodesic convex hulls. In the following consider a polygonal domain P with n vertices. For two points $x, y \in P$ denote by $\Pi_P(x, y)$ the set of *geodesics* between x and y in P. That is, every element of $\Pi_P(x, y)$ is a curve from x to y that is contained in P and corresponds to a shortest path between x and y (among all curves between x and y in P). A set $K \subseteq P$ is *geodesically convex* (in P) if, for every $x, y \in K$, all geodesics in $\Pi_P(x, y)$ are contained in K. For $S \subseteq P$, the *geodesic convex hull*, or simply G*-hull* of S in P, is the (inclusion) minimum geodesically convex set $\mathsf{GH}_P(S) \subseteq P$ that contains S. In this paper, we study the case in which S consists of a finite set of $O(n)$ points (called *sites*).

One way to conceive the G-hull of S is to start with $C_0 = S$ and iteratively add more points as follows. In the i-th step, for every pair of points $x, y \in C_{i-1}$ (possibly infinitely many) take all geodesics in $\Pi_P(x, y)$ and add them to the new set C_i. Continue until $C_i = C_{i-1}$ at the end of some step. Note that this procedure as described is not an algorithm because (i) the number of pairs/geodesics to consider is not finite in general and (ii) it is not clear whether the procedure terminates after a finite number of steps.

Visibility graphs and shortest path maps. Every geodesic in $\Pi_P(x, y)$, for $x, y \in S$, forms a path in the *visibility graph* $\mathrm{Vis}_P(S)$ of S with respect to P. This graph is defined on the vertex set $V = S \cup \mathsf{V}(P)$ and two vertices $x, y \in V$ are *visible* and connected by an edge in $\mathrm{Vis}_P(S)$ if the relative open line segment $\overline{xy} \setminus \{x, y\}$ is contained in $P \setminus V$. For given P and S, the graph $\mathrm{Vis}_P(S)$ can be computed in $O(|V|^2)$ time and space [22].

For a point $s \in P$, the *shortest path map* (SPM) for s is the subdivision of P into *cells* to which the geodesic from s passes through the same sequence of vertices of P. There are $O(n)$

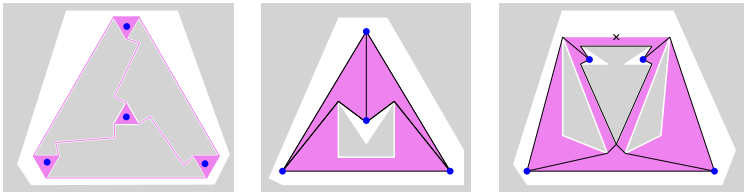
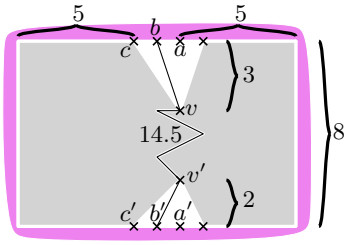


Figure 2 No site appears in the boundary (left). In the middle figure there is no way of partitioning the four sites so that the convex hulls of the two sets intersect. To the right, the top point (cross) belongs to the G-hull of the four sites, but it is not included in the G-hull of any three sites. The geodesics between pairs of sites are shown in black.



$$
\begin{aligned}
d(v, v') &= 14.5 \\
d(a, a') &= 18 < d(a, v) + d(v, v') + d(v', a') = 19.5 \\
d(b, b') &= \sqrt{5} + \sqrt{10} + 14.5 \approx 19.79 \\
&< 20 = d(b, a) + d(a, a') + d(a', b') \\
d(a, c') &= 20 < d(a, v) + d(v, v') + d(v', c') \approx 20.33 \\
d(a', c) &= 20 < d(a', v') + d(v, v') + d(v, c) \approx 20.1 \\
d(a, b') &= 19 < d(a, v) + d(v, v') + d(v', b') \approx 19.73 \\
d(a', b) &= 19 < d(a', v') + d(v, v') + d(v, b) \approx 19.66
\end{aligned}
$$

Figure 3 A partial drawing of a domain (violet), in which eight points have been highlighted. The geodesic between b and b' passes through v and v', while the geodesics between pairs a and a' or c and c' do not pass through v or v'. The function $d(\cdot, \cdot)$ denotes the geodesic distance.

such cells, and the boundaries between these cells are formed by curves of constant algebraic degree. Hershberger and Suri [15] provide an $O(n \log n)$ time algorithm to construct the SPM for a given point s. Given the SPM for s, we can compute the geodesic distance from s to any query point $p \in P$ in $O(\log n)$ time using point location. In the same time, we can also get the first and last vertex (other than s and p, if any) of some path in $\Pi_P(s, p)$.

Remarkable properties of g-hulls. Figure 2 depicts a polygon P with all sites in the interior of $\mathsf{GH}_P(S)$, as well as an example where an analogue of Radon's Theorem does not hold, i.e., there is no partition of S into two non-empty sets $S_1 \cup S_2$ such that $\mathsf{GH}_P(S_1) \cap \mathsf{GH}_P(S_2) \neq \emptyset$. Similarly, Figure 2 (right) depicts an example where the natural extension of Carathéodory's Theorem does not hold: there exists a point in $\mathsf{GH}_P(S)$ that does not belong to the G-hull of any three sites of S. An example where the actual distance between points influences the structure of the G-hull is given in the (partial) instance depicted in Figure 3. Moving the points slightly without changing the order type can have large influence on the structure of the G-hull.

3 Cactus domains and general properties

Even in a single simple polygon, the G-hull of two segments on its boundary forms a so-called funnel [13], which, in general, is not simple. It is therefore natural to study a slightly more general class of polygons to be able to describe G-hulls. A frequently used relaxation is referred to as a weakly simple polygon, which, intuitively speaking, allows the curve that describes the boundary to touch but not properly cross itself. However, to make this intuition formally precise is surprisingly cumbersome [8]. For describing G-hulls, a much more restricted class of polygons is sufficient, which we will define in the next paragraphs. (We refrain from using the term "weakly simple polygon" to emphasize this difference.)

In a *plane drawing* or *embedding* of a connected graph, the vertices are represented by pairwise distinct points, edges are represented by Jordan arcs connecting their endpoints, and no two edges intersect except at a common endpoint. In a *straight-line* drawing, all Jordan arcs representing edges are (straight) line segments. A *cactus* is a connected graph in which every edge belongs to at most one simple cycle. Cacti are *outerplanar*, that is, they can be embedded in the plane so that all vertices are incident to one particular face (which is usually the outer face).

A *cactusgon* is a domain K that is represented by an outerplane straight-line drawing $\varphi(G)$ of a cactus G: The interior $\mathsf{int}(K)$ is formed by the union of all (open) bounded faces in $\varphi(G)$, the boundary ∂K is formed by the union of all edges in $\varphi(G)$, and collectively $K = \mathsf{int}(K) \cup \partial K$. We obtain a combinatorial representation of ∂K as the unique circular sequence of edges and vertices as they appear along the boundary of the outer face of $\varphi(G)$ in counterclockwise order. A closed curve that traverses ∂K of some cactusgon K in this fashion is a *cactus curve*. Note that K is a compact subset of $\mathbb{R}^2$ and that every simple polygon is a cactusgon whose associated graph G is a simple cycle. Consider a face of a cactus that is incident to all vertices (which may or may not be unbounded); we call this open subset of the plane a *cactus face*. The boundary of a cactus face is defined analogously to the one of a cactusgon (but note that the boundary is not part of the cactus face).

▶ **Definition 2.** A *cactus domain* or, for short, *C-domain* K is a planar region bounded by an *outer polygon* $\mathcal{C}$ and $t \geq 0$ *inner voids* $K_1, \ldots, K_t$, where (1) $\mathcal{C}$ is a cactusgon, (2) $K_i \subseteq \mathcal{C}$ is a bounded cactus face, for $1 \leq i \leq t$, and $K_1, \ldots, K_t$ are pairwise disjoint. The *outer void* K_0 of K is is an unbounded cactus face (or, equivalently, the outer face of $\mathcal{C}$). The interior of K is $\mathsf{int}(K) = \mathbb{R}^2 \setminus \bigcup_{i=0}^{t}(K_i \cup \partial K_i)$, the boundary of K is $\partial K = \bigcup_{i=0}^{t} \partial K_i$, the vertices of K are $\mathsf{V}(K) = \bigcup_{i=0}^{t} \mathsf{V}(K_i)$, and collectively $K = \mathsf{int}(K) \cup \partial K$.

Note that the above definition slightly abuses notation since $V(\cdot)$ was only defined for polygons. Along the paper we do a similar abuse for structures defined for polygons (such as shortest path map and visibility graph) and apply them to cactusgons. The extension of these concepts (and the algorithims) are straighforward. Thus, for simplicity we omit them.

Observe that $\partial\mathcal{C} = \partial K_0$ and so $\mathsf{V}(\mathcal{C}) = \mathsf{V}(K_0)$. Again, we write $K = (\mathcal{C}, K_1, \ldots, K_t)$ as a shorthand. As usual, $\mathsf{int}(K)$ is an open subset of $\mathbb{R}^2$ and K is a compact subset of $\mathbb{R}^2$ with $\mathbb{R}^2 \setminus K = \bigcup_{i=0}^{t} \mathsf{int}(K_i)$. Observe that the cycles of K may share edges or points of their boundary. As an extreme example, if $\mathcal{C}$ is simple, then we may even have $t = 1$ and one large hole $K_1 = \mathcal{C}$ (in this case, K is just a one-dimensional polygonal cycle). While the theorem below is not hard to prove in a stand-alone way, it will follow from our algorithmic construction of G-hulls, as our algorithm produces a C-domain that we prove to coincide with the G-hull of S.

▶ **Theorem 3.** *Given a polygonal domain P with n vertices in total and a set $S \subset P$ of $O(n)$ sites, the geodesic convex hull $\mathsf{GH}_P(S)$ of S in P is a cactus domain whose vertices are from $S \cup \mathsf{V}(P)$ and whose edges are edges of the visibility graph $\mathrm{Vis}_P(S)$. In particular, the boundary of $\mathsf{GH}_P(S)$ can be described as a plane straight-line graph on $O(n)$ vertices.*

4 Characterization of geodesically convex sets

The aim of this section is to give a characterization of C-domains that are geodesically convex in a polygonal domain $P = (P_0, \ldots, P_h)$. Consider a C-domain $K = (\mathcal{C}, K_1, \ldots, K_t)$. If K is not geodesically convex, then there exist two points $p, q \in K$ and a geodesic $\pi \in \Pi_P(p, q)$ such that $\pi \not\subset K$. For simplicity, assume that $\pi \cap K = \{p, q\}$. (That is, the geodesic only

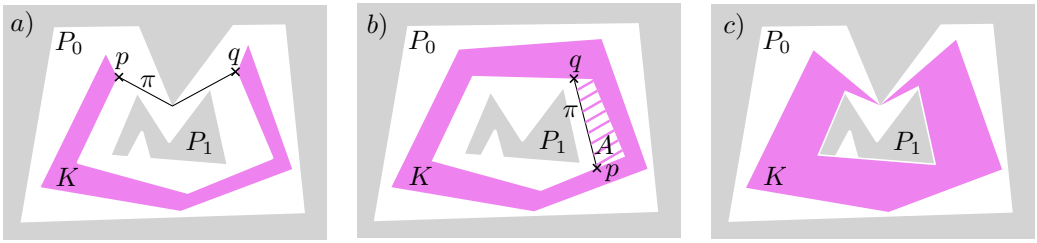


Figure 4 *a*) A C-domain K that is divisible by π. *b*) An indivisible C-domain K that is not tight. *c*) An indivisible and tight C-domain K.

touches K at the endpoints. This can be achieved by restricting π.) As $K_0, \ldots, K_t$ are pairwise interior-disjoint, p and q lie in the same component of ∂K, say ∂K_i. Therefore, π splits the void K_i into two parts A and B; refer to Figure 4 for illustrations.

In the case in which one of the two parts, say A, contains no hole of P and also not its outer face (Figure 4b), we use a local operation to enlarge K following the rubber band metaphor. We show in Lemma 4 that all of A is in $\mathsf{GH}_P(K)$. A C-domain without such a geodesic is called tight.

The other possible situation is that both A and B contain at least one hole or the outer face of P. In this case we have a path that is topologically different from all paths in K, and we say that K is divisible by π (Figure 4a). If no such path exists, then K is indivisible.

Clearly, any geodesically convex C-domain must be indivisible and tight. In the remainder of this section, we prove in form of a characterization that the reverse implication holds as well, i.e., a C-domain K is geodesically convex if and only if it is indivisible and tight.

Tightness of cactus domains. For $1 \leq i \leq t$, let $\mathsf{V}_K(i) = \bigcup_{P_j \subseteq K_i} \mathsf{V}(P_j)$ be the vertices of all holes P_j of P for which P_j is contained in the void K_i. Observe that, in general, ∂K_i may contain vertices of holes of P not contained in K_i. Thus, $\mathsf{V}_K(i)$ may be different from the set of vertices of P contained in K_i. As P_0 is not contained in any inner void of K, for the outer void, we let $\mathsf{V}_K(0) = \mathsf{V}(P_0) \cup \bigcup_{P_j \subseteq K_0} \mathsf{V}(P_j)$. In particular, $\mathsf{V}_K(0) \neq \emptyset$.

A curve γ *separates* two compact subsets $A, B \subset \mathbb{R}^2$ if every curve that connects a point in A with a point in B intersects γ. Given a void K_i of K with $\mathsf{V}_K(i) \neq \emptyset$, we define the *reduction* $\varrho(K_i)$ as the minimum length curve in P that separates $\mathsf{V}_K(i)$ from $\mathsf{int}(K)$ (possibly $\varrho(K_i) = \partial K_i$). We can think of $\varrho(K_i)$ as being obtained by continuously tightening a curve tracing ∂K_i as much as possible while maintaining separation between $\mathsf{V}_K(i)$ and K.

Algorithmically, an inner void K_i, $i \geq 1$, can usually be treated as a simple polygon. It follows from Toussaint's algorithm [20] that $\varrho(K_i)$ is a (non-simple, in general) closed walk in $\mathrm{Vis}_{K_i}(\mathsf{V}_K(i))$; in fact, $\varrho(K_i)$ is a cactus curve. Similarly, for the outer void K_0 the outside domain is formed by the outer void and a collection of simple polygons (P_0 and possibly some holes in the exterior of K). The algorithm of de Berg [10] asserts that $\varrho(K_0)$ is a cactus curve in this case as well.

For an inner void K_i, $i \geq 1$, the boundary ∂K_i encloses $\varrho(K_i)$. For the outer void K_0, the curve $\varrho(K_0)$ encloses ∂K_0; see Figure 5. Regardless, ∂K_i and $\varrho(K_i)$ form an annulus (possibly with no interior point). We say that any point in this annulus lies *between* ∂K_i and $\varrho(K_i)$. Given a C-domain K, a void K_i of K is *tight* if $\mathsf{V}_K(i) \neq \emptyset$ and $\partial K_i = \varrho(K_i)$. We say that K is *tight* if K_i is tight for each $0 \leq i \leq t$. If K is tight, then for each $1 \leq i \leq t$, the void K_i contains at least one hole of P; otherwise, $\mathsf{V}_K(i)$ would be empty.

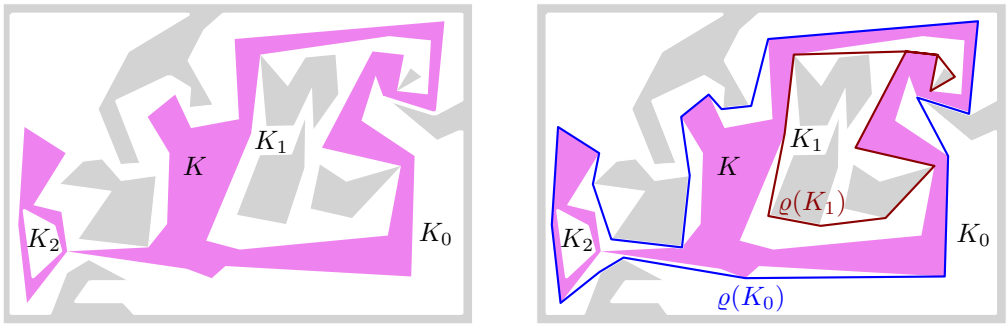


Figure 5 Left: A C-domain K with two inner voids. Right: The reductions of the voids K_0 and K_1 are depicted. The curve $\varrho(K_1)$ is enclosed by ∂K_1, while ∂K_0 is enclosed by $\varrho(K_0)$ (curves shown in solid red and blue, respectively). Every point between ∂K_i and $\varrho(K_i)$ belongs to $\mathsf{GH}_P(K)$. Notice that, since K_2 has no hole, we have $K_2 \subseteq \mathsf{GH}_P(K)$.

▶ **Lemma 4.** *Let P be a polygonal domain, $K = (\mathcal{C}, K_1, \ldots, K_t)$ a C-domain in P, and K_i a void of K, for $0 \leq i \leq t$. (1) If $V_K(i) \neq \emptyset$, then each point that lies between ∂K_i and $\varrho(K_i)$ belongs to $GH_P(K)$. (2) If $V_K(i) = \emptyset$, then $K_i \subseteq GH_P(K)$.*

Characterization of geodesically convex cactus domains. Using the above lemma, we are ready to give sufficient and necessary conditions for a C-domain to be geodesically convex. It remains to formally define divisibility. Given a void K_i, we say that two points $p, q \in \partial K_i$ *separate* K_i if a geodesic in $\Pi_P(p, q)$, called *separating geodesic*, splits K_i into two connected components, each containing either at least one hole of P or the outer face of P; see Figure 4a. We say that K_i is *divisible* if some pair of points on ∂K_i separates K_i. Analogously, K_i is *indivisible* if no pair of points on ∂K_i separates K_i. A C-domain is *divisible* if at least one of its voids is divisible; otherwise, it is *indivisible*.

▶ **Theorem 5.** *A C-domain K is geodesically convex if and only if it is indivisible and tight.*

Theorem 5 is the main structural result our algorithm relies on. To algorithmically test divisibility of a C-domain, we use the following lemma.

▶ **Lemma 6.** *If a C-domain K is divisible, then there exists a geodesic π separating a void K_i with the following three properties. (i) The intersection of π with ∂K consists only of its endpoints. (ii) It contains a vertex of P in its relative interior or both of its endpoints are vertices of K_i. (iii) It consists of at least one segment that intersects the interior of K_i.*

5 Computing the geodesic convex hull

In this section, we present an algorithm that, given a polygonal domain P and a set S of sites, computes $\mathsf{GH}_P(S)$. To simplify the presentation and the analysis, we assume that $|S| = O(n)$, where n is the number of vertices of P. But the exact dependency on $|S|$ can be easily derived from our proofs.

Our algorithm is founded upon the characterization of Theorem 5. We first start with the C-domain formed by the union of all geodesics going from an arbitrary site of S to all other sites. As a next step, we make use of Lemma 4 on the resulting C-domain to obtain a new tight C-domain that we test for divisibility. If this tight C-domain is divisible, our procedure reports a geodesic that separates it, which we add to the C-domain. The addition of this separating geodesic generates a new C-domain that is not necessarily tight. We repeat the procedure iteratively until we obtain a C-domain that is both tight and indivisible. Then by Theorem 5 this domain is $\mathsf{GH}_P(S)$.

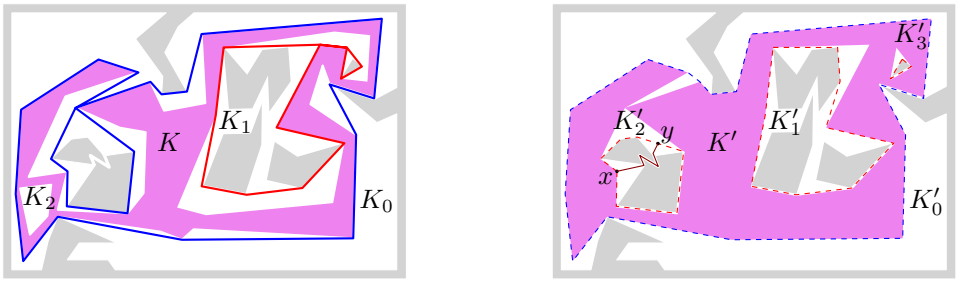


Figure 6 Left: A C-domain K and the reduction of each of its voids. Right: The new C-domain K' obtained from K after applying steps (1) and (2) of the tightening process. In the left figure we have two inner voids. The reduction of K_0 and K_1 are shown in solid blue and red curves, respectively (K_2 need not be reduced because it has no holes). The right figure shows the resulting C-domain K'. This domain is not tight, so step (2) needs to be applied a second time (the region to be added is shown dashed). Note that the indivisible void K'_2 may become divisible after the tightening process (due to the geodesic between points x and y).

Computing tightenings of cactus domains. We introduce the *tightening process* of a C-domain $K = (\mathcal{C}, K_1, \ldots, K_t)$. Intuitively speaking, we want to enlarge K as little as possible until it is tight. The result is another C-domain K', which we call the *tightening* of K. In order to do so, we proceed as follows: (1) for each $1 \leq i \leq t$ such that $\mathsf{V}_K(i) = \emptyset$, we add each point in this void to the tightening of K (effectively removing this void from K), and (2) for each $0 \leq i \leq t$ such that $\mathsf{V}_K(i) \neq \emptyset$, compute the reduction of K_i and add the space in the annulus between ∂K_i and $\varrho(K_i)$ to the tightening of K. Recall that the reduction $\varrho(K_i)$ of a void K_i of K need not be a simple curve. Therefore, to obtain a valid C-domain, we consider each bounded component of $\mathbb{R}^2 \setminus \varrho(K_i)$ (which are cactus faces) and add them as new voids replacing K_i. In particular, the resulting C-domain may have more voids than K (and they need not be tight, see Figure 6). Thus, we apply again step (2) iteratively until we obtain a C-domain in which the boundary of each void coincides with its reduction. Since every newly created void needs to contain a hole of P, we obtain this C-domain after at most h iterations. Since in the resulting domain the boundary of each void coincides with its reduction, we obtain a tight C-domain K', the tightening of K.

▶ **Lemma 7.** *Given a C-domain K with $O(n)$ vertices, we can compute the tightening of K in $O(hn \log n)$ time. Moreover, the tightening of K is a C-domain whose edges belong to $\mathrm{Vis}_P(\mathsf{V}(K))$.*

Testing divisibility of cactus domains. In this section we provide a deterministic algorithm to determine if a C-domain is divisible. This property is considerably harder to test than tightness. In fact, this test is the main bottleneck of our algorithm and the main algorithmic challenge of this paper.

Let $K = (\mathcal{C}, K_1, \ldots, K_t)$ be a tight C-domain. To test the divisibility of K, we test each void K_i separately. Using Lemma 6, it is sufficient to determine whether there is a separating geodesic containing a vertex of P in its relative interior or that is a segment between two vertices of K_i that see each other. The latter can be easily tested using the visibility graph of P. For testing the former, we modify an algorithm by Bae and Okamoto [5]: this $O(n^{5+\varepsilon})$-time algorithm takes a polygonal domain on n vertices, and encodes all geodesics between pairs of points on its boundary as the lower envelope of a collection of constant-degree distance functions. While their algorithm serves to construct a data structure for shortest-path queries

among points on the boundary of a polygonal domain, we are able to translate its main ideas to test divisibility. Additionally, several new observations allow us to replace a factor of $O(n^{2+\varepsilon})$ for a factor of $O(h^{2+\varepsilon})$ in the running time. The remaining part of this section describes our algorithm in detail.

As a preprocessing step, compute the SPM from every vertex of P in overall $O(n^2 \log n)$ time [15]. Then, for each edge e of K_i, split e at each point of intersection with the boundary of a cell in the SPM of some vertex of P. In this way, we obtain the *spm-subdivison* of e into *spm-segments*. The *spm-subdivision* of ∂K_i is the union of the spm-subdivision of its edges.

Let $\text{SPM}(p)$ be the SPM for a point p. We claim that if, for some vertex v of P and cell c of $\text{SPM}(v)$, c intersects ∂K_i in three or more connected components, then there is a segment t contained in this cell connecting two points of ∂K_i through the interior of K_i. Moreover, t must be a separating geodesic, as otherwise t would split K_i into two components, one of which would not contain a vertex of $\mathsf{V}_K(i)$. However, since t can be used as a shortcut to reduce the length of ∂K_i while separating $\mathsf{V}_K(i)$ from K, we obtain a contradiction with the tightness of K_i, which proves our claim. Thus, if a cell of the SPM of some vertex of P intersects ∂K_i in three or more connected components, then K_i is divisible.

Therefore, to compute the spm-subdivision, we first compute the intersection points of the SPM of each vertex with ∂K_i, and then sort all these intersection points along the boundary of K_i to obtain the spm-subdivision of ∂K_i. If at some point during this process we find a cell of a SPM that intersects ∂K_i in more than two connected components, then the algorithm finishes and reports the separating geodesic contained in this cell. Thus, we assume from now on that no cell of an SPM intersects ∂K_i in more than two connected components, i.e., each cell of an SPM contributes to $O(1)$ spm-segments to the spm-subdivision. Because we consider the SPM of the n vertices of P, each with $O(n)$ cells, the spm-subdivision of K consists of $O(n^2)$ spm-segments, and the total running time of our preprocessing step is bounded by $O(n^2 \log n)$.

An important property of the spm-subdivision is that for a spm-segment s and a point $x \in s$, the set of vertices of P that are visible from x remains unchanged as x moves along s. Thus, we let V_s be the set of vertices of P visible from s. For a pair of spm-segments s and s', each geodesic with at least two segments from a point in s to a point in s' *starts* with a vertex v in V_s (i.e., v is the first vertex visited by this path after leaving s). Moreover, because s' is contained in a single cell of $\text{SPM}(v)$, a geodesic from v to any point in s' must have the same combinatorial structure. Let v^* be the last vertex visited in the path from v to any point of s' (note that we may have $v = v^*$). Then, any path from a point $x \in s$ to a point $y \in s'$ can be parametrized by the distance function $f_v(x, y) = d(x, v) + d(v, v^*) + d(v^*, y)$. Because $d(v, v^*)$ is a known constant, f_v is a constant degree algebraic function from $s \times s'$ to $\mathbb{R}$. We could then compute the minimization diagram of the set $F_{s,s'} = \{f_v(x, y) : v \in V_s\}$, i.e., the lower envelope of these distance functions over all different starting vertices. This diagram has the following property: the algebraic surface patch of f_v appears in this lower envelope if and only if there is a geodesic from a point $x \in s$ to a point $y \in s'$ that passes through v. We now look for a vertex v that lies in the interior of K_i and f_v appears in the lower envelope. If this happens, there is a separating geodesic connecting s with s' starting at v (and thus we conclude that K_i is divisible); note that by Lemma 6 this is sufficient to determine divisibility. This gives us an algorithm to decide divisibility whose running time is dominated by the computation of $O(n^4)$ minimization diagrams, one for each pair of spm-segments. We will improve this later but first, we look in more detail at the starting vertices of the geodesics we need to consider. The following observation leads to our main improvement when compared to the algorithm of Bae and Okamoto [5].

▶ **Lemma 8.** *Given an spm-segment s and a hole H of P, there are at most two starting vertices in H among all geodesics going from a point in s to a point in ∂K_i. Moreover, they are the counterclockwise- and clockwise-most vertices in $V_s \cap V(H)$, when sorted radially around any point in s.*

Therefore, at most two geodesics from s to ∂K_i can start at different vertices of $V_s \cap V(H)$. That is, each hole can contribute to at most two start vertices, hence only a total of $O(h)$ starting vertices must be considered.

By Lemma 8, we can let $V_s^* \subseteq V_s$ denote the set of $O(h)$ starting vertices of paths going from s to ∂K_i. Moreover, we can compute V_s^* in $O(n)$ time by computing the maximum and minimum element, among the vertices of each hole of P, in the radial order around an arbitrary point of s. Because we need to consider only $O(h)$ vertices in V_s^*, we notice that there are many divisions among spm-segments that do not correspond to the boundary of a cell in the SPM of a vertex in V_s^*. Thus, we could modify our spm-subdivision with respect to s and consider only the breaking points induced by the SPM of a vertex in V_s^*. Because each SPM has complexity $O(n)$ and since $|V_s^*| = O(h)$, this induces at most $O(nh)$ divisions. In this way, we obtain a partition of ∂K_i into $O(nh)$ *s-segments*, each being a collection of consecutive spm-segments. The idea of using this subdivision is that, to compute a minimization diagram of distance functions between s and an s-segment, we need to consider only $O(h)$ functions defined by the vertices in V_s^*.

▶ **Theorem 9.** *We can determine if a tight C-domain K of $O(n)$ vertices in a polygonal domain $P = (P_0, \ldots, P_h)$ of n vertices is divisible in $O(n^3 h^{2+\varepsilon})$ time.*

Proof. Let s be a spm-segment. Note that, when going from one s-segment to a neighboring one, the SPM cell of at most one vertex in V_s^* can change. Intuitively, this means that the distance functions we need to consider have "little" variation among neighboring s-segments. We formalize this intuition as follows. Group h consecutive s-segments lying on the same edge of ∂K_i and take their union to produce an *s-block* g. We claim that $O(h)$ distance functions need to be considered to compute the minimization diagram encoding all geodesics from s to any s-block g. To show this, for each $v \in V_s^*$, let τ_v be the number of cells of $\textsc{spm}(v)$ that intersect s-block g. Let σ be a cell of $\textsc{spm}(v)$ that intersects g. Notice that there is exactly one ending vertex v^* in any geodesic from v to $\sigma \cap g$. Thus, we can define an *s-g-function* $f_{v,\sigma} : s \times (\sigma \cap g) \to \mathbb{R}$ such that $f_{v,\sigma}(x, y) = d(x, v) + d(v, v^*) + d(v^*, y)$. Note that there are exactly τ_v s-g-functions defined for each vertex v of V_s^*. Because g consists of h s-segments, we know that g can be intersected by at most $O(h)$ cells among the SPMs of the vertices in V_s^*. Therefore, $\sum_{v \in V_s^*} \tau_v = O(h)$, i.e., there are in total $O(h)$ s-g-functions defined for all vertices of V_s^*. Moreover, any geodesic from s to g needs to start with a vertex of V_s^* and hence, it is considered in one of these functions. Consequently, the minimization diagram of the s-g-functions encodes the distance of all geodesics going from s to g. Note that this minimization diagram can be computed in $O(h^{2+\varepsilon})$ time [19]. After computing it, we can check within the same time whether there is a geodesic between s and g that goes through the interior of K_i by going through all elements of this lower envelope.

By grouping all $O(nh)$ s-segments into consecutive s-blocks of at most h spm-segments, each contained in a single edge of ∂K_i, we obtain $O(n)$ s-blocks in total along ∂K_i. Therefore, we need to compute $O(n)$ minimization diagrams for a given spm-segment s, one for each s-block, each in $O(h^{2+\varepsilon})$ time. Repeating this over all $O(n^2)$ spm-segments gives a total running time of $O(n^3 h^{2+\varepsilon})$. ◀

▶ **Theorem 1.** *Let P be a polygonal domain with n vertices and h holes, and let $S \subset P$ be a set of $O(n)$ sites. The geodesic convex hull of S in P can be computed in $O(n^3 h^{3+\varepsilon})$ time.*

Proof. Let s be a site of S. For each $s' \in S \setminus \{s\}$, choose an arbitrary path in $\Pi_P(s, s')$. Let K^0 be the plane connected graph obtained by taking the union of all chosen paths. Notice that K^0 is a connected C-domain that contains all sites of S. Moreover, because K^0 is plane (as no two geodesics from s can cross), K^0 consists of $O(n)$ vertices and edges and $K^0 \subseteq \mathsf{GH}_P(S)$ (as it consists of geodesics between points of S). We describe now a recursive procedure that incrementally constructs the G-hull of S starting from K^0.

Given a C-domain K^r for some even number r, we construct K^{r+1} as the tightening of K^r using Lemma 7 in $O(hn \log n)$ time. Since P has h holes, K^r is a tight C-domain with at most h voids whose vertices and edges are contained in $\mathrm{Vis}_P(S)$. Thus, because K^{r+1} is plane, it has complexity $O(n)$. We then use Theorem 9 to test whether K^{r+1} is divisible or not, which takes $O(n^3h^{2+\varepsilon})$ time. If K^{r+1} is indivisible, then as it is also tight, Theorem 5 implies that K^{r+1} is geodesically convex. Thus, as $S \subset K^{r+1}$ and since $\mathsf{GH}_P(S)$ is the smallest geodesically convex set that contains S, we get that $\mathsf{GH}_P(S) \subseteq K^{r+1}$. Moreover, because all points in K^{r+1} belong to $\mathsf{GH}_P(S)$ by Lemma 4, we know that $K^{r+1} \subseteq \mathsf{GH}_P(S)$. Therefore, if K^{r+1} is indivisible, then $K^{r+1} = \mathsf{GH}_P(S)$ and we are done.

Otherwise K^{r+1} is divisible and we have found a separating geodesic, i.e., there is some void of K^{r+1} and two points x and y such that the path $\pi_P(x, y)$ separates K^{r+1}. In this case, we add the path $\pi_P(x, y)$ to K^{r+1} and obtain a new C-domain $K^{r+2} \subset \mathsf{GH}_P(S)$ that is not necessarily tight. Because $r + 2$ is even, we can repeat this procedure recursively until finding a tight indivisible C-domain. One may think that one test for divisibility suffices, i.e., that this does not need to be repeated every time that a tightening is computed. However, the tightening of an indivisible C-domain may be divisible; see Figure 6.

Note that in each round, if the tight C-domain K^{r+1} is divisible, then we find a new separating geodesic that separates two holes of P that were previously in the same void of K^{r+1}. In particular, we create a new void with at least one hole. Since we can have at most h such voids, the above procedure will iterate at most h times and must end with a tight indivisible domain that coincides with $\mathsf{GH}_P(S)$.

The running time is dominated by the divisibility test given by Theorem 9 which has to be executed at most h times. Thus, the total running time becomes $O(n^3h^{3+\varepsilon})$ as claimed. ◀

References

1 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete Comput. Geom.*, 56(4):836–859, 2016.

2 Oswin Aichholzer, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic order types. *Algorithmica*, 70(1):112–128, 2014. doi:10.1007/s00453-013-9818-8.

3 Sang Won Bae, Matias Korman, and Yoshio Okamoto. The geodesic diameter of polygonal domains. *Discrete Comput. Geom.*, 50(2):306–329, 2013.

4 Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. In *Proc. 26th Canadian Conf. on Computational Geometry*, 2014.

5 Sang Won Bae and Yoshio Okamoto. Querying two boundary points for shortest paths in a polygonal domain. *Comput. Geom.*, 45(7):284–293, 2012. doi:10.1016/j.comgeo.2012.01.012.

6 Ahmad Biniaz, Prosenjit Bose, Anil Maheshwari, and Michiel H. M. Smid. Plane geodesic spanning trees, hamiltonian cycles, and perfect matchings in a simple polygon. *Comput. Geom.*, 57:27–39, 2016. doi:10.1016/j.comgeo.2016.05.004.

7 Prosenjit Bose, Erik D. Demaine, Ferran Hurtado, John Iacono, Stefan Langerman, and Pat Morin. Geodesic ham-sandwich cuts. *Discrete & Computational Geometry*, 37(3):325–339, 2007. doi:10.1007/s00454-006-1287-2.

8 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1655–1670. SIAM, 2015. doi:10.1137/1.9781611973730.110.

9 Yi-Jen Chiang and Joseph S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, 1999.

10 Mark de Berg. Translating polygons with applications to hidden surface removal. In John R. Gilbert and Rolf G. Karlsson, editors, *SWAT 90, 2nd Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 11-14, 1990, Proceedings*, volume 447 of *Lecture Notes in Computer Science*, pages 60–70. Springer, 1990. doi:10.1007/3-540-52846-6_78.

11 Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Henk Meijer, Mark H. Overmars, and Sue Whitesides. Separating point sets in polygonal environments. *Int. J. Comput. Geometry Appl.*, 15(4):403–420, 2005. doi:10.1142/S0218195905001762.

12 Peter Eades and David Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14(9):715–718, 1993. doi:10.1016/0167-8655(93)90140-9.

13 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

14 John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM J. Comput.*, 26(6):1612–1634, 1997.

15 John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999. doi:10.1137/S0097539795289604.

16 Anna Lubiw, Daniela Maftuleac, and Megan Owen. Shortest paths and convex hulls in 2d complexes with non-positive curvature. *CoRR*, abs/1603.00847, 2016. arXiv:1603.00847.

17 Joseph S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry, Second Edition*. Chapman and Hall/CRC, 2004.

18 Joseph S. B. Mitchell, Günter Rote, and Gerhard J. Woeginger. Minimum-link paths among obstacles in the plan. *Algorithmica*, 8(5&6):431–459, 1992. doi:10.1007/BF01758855.

19 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12(3):327–345, 1994.

20 Godfried T. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3(2):9–42, 1989.

21 Haitao Wang. On the geodesic centers of polygonal domains. In *Proc. 24rd Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, pages 77:1–77:17, 2016.

22 Emo Welzl. Constructing the visibility graph for n-line segments in $o(n^2)$ time. *Inf. Process. Lett.*, 20(4):167–171, 1985. doi:10.1016/0020-0190(85)90044-4.

Tree Containment With Soft Polytomies

Matthias Bentert
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
matthias.bentert@tu-berlin.de

Josef Malík
Czech Technical University, Prague, Czech Republic
josef.malik@fit.cvut.cz

Mathias Weller
CNRS, LIGM, Université Paris Est, Marne-la-Vallée, France
mathias.weller@u-pem.fr

Abstract

The Tree Containment problem has many important applications in the study of evolutionary history. Given a phylogenetic network N and a phylogenetic tree T whose leaves are labeled by a set of taxa, it asks if N and T are consistent. While the case of binary N and T has received considerable attention, the more practically relevant variant dealing with biological uncertainty has not. Such uncertainty manifests itself as high-degree vertices ("polytomies") that are "jokers" in the sense that they are compatible with any binary resolution of their children. Contrasting the binary case, we show that this problem, called Soft Tree Containment, is $\mathcal{NP}$-hard, even if N is a binary, multi-labeled tree in which each taxon occurs at most thrice. On the other hand, we reduce the case that each label occurs at most twice to solving a 2-SAT instance of size $O(|T|^3)$. This implies $\mathcal{NP}$-hardness and polynomial-time solvability on reticulation-visible networks in which the maximum in-degree is bounded by three and two, respectively.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Applied computing → Biological networks

Keywords and phrases Phylogenetics, Reticulation-Visible Networks, Multifurcating Trees

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.9

Acknowledgements The project leading to this work was conceived on the 2017 research retreat of the Algorithmics and Computational Complexity group of TU Berlin.

1 Introduction

With the dawn of molecular biology also came the realization that evolutionary trees, which have been widely adopted by biologists, are insufficient to describe certain processes that have been observed in nature. In the last decade, the idea of reticulate evolution, supporting gene flow from multiple parent species, arose [2, 15]. A reticulation event can be caused by, for example, hybridization (occurring frequently in plants) and horizontal gene transfer (a dominating factor in bacterial evolution). Reticulate evolution is described using "phylogenetic networks" (see the monographs by Gusfield [11] and Huson et al. [13]). A central question when dealing with both phylogenetic trees and networks is whether or not they represent consistent information, formulated as the question whether or not the network "displays" the tree. This problem is known as Tree Containment and it has been shown $\mathcal{NP}$-hard [14, 17]. Due to its importance in the analysis of evolutionary history, attempts have been made to identify polynomial-time computable special cases [6, 5, 1, 10, 14, 17, 7, 18],

as well as moderately exponential-time algorithms [8, 18]. However, all of these works are limited to binary networks and trees.

In reality, we cannot hope for perfectly precise evolutionary histories. In particular, speciation events (a species splitting off another) occurring in rapid succession (only a few thousand years between speciations) can often not be reliably placed in the correct order they occurred. The fact that the correct order of bifurcations is unknown is usually modeled by multifurcating vertices and, to tell them apart from speciation events resulting in multiple species, the former are called "soft polytomies" and the latter are called "hard polytomies". Of course, the same argument holds for non-binary reticulation vertices indicating uncertainty in the order of hybridization events. Soft polytomies have a noteworthy impact on the question of whether a tree is compatible with a network: since a soft polytomy (also called "fan") on the taxa a, b, and c represents lack of knowledge regarding their history, we would consider any binary tree on the taxa a, b, and c compatible with it. In this work, we present first algorithmic results for Tree Containment with soft polytomies (which we call Soft Tree Containment). We consider the case where the network is a multi-labeled tree and show that the problem is cubic-time solvable if each label occurs at most twice (by reduction to 2-SAT) and $\mathcal{NP}$-hard, otherwise. This implies corresponding results for (single-labeled) "reticulation-visible" networks, depending on their maximum in-degree. Despite being an intermediate step in proving results for networks, multi-labeled trees are themselves important, for example when handling gene trees, in which different versions of a gene may be found in the same species.

Finally, our results have impact on the Cluster Containment problem [13] since it is a special case of our problem.[1]

Preliminaries

A *phylogenetic network* (or *network* for short) on a set X of taxa is a rooted, leaf-labeled DAG in which all vertices that do not have in-degree at most one have out-degree exactly one. These vertices are called *reticulations* and the others are called *tree vertices*. A network without reticulations is called a (phylogenetic) *tree*. By default, no label occurs twice in a network, and we will make exceptions explicit by calling networks in which a label may occur more than once *multi-labeled* (note that networks are a special case of multi-labeled networks in which each label occurs only once). This allows us to use leaves and labels (taxa) interchangeably. For brevity, we abbreviate $\{x, y\}$ to xy, and $\{x, y, z\}$ to xyz. Let N be a network with root ρ_N. We denote the set of vertices in N by $V(N)$. We define a relation "$\leq_N$" on subsets of $V(N)$ such that $U \leq_N W$ if and only if N contains a w-u-path for each $u \in U$ and $w \in W$. If $u \leq_N w$, we call u a *descendant* of w and w an *ancestor* of u. For each $v \in V(N)$, we let N_v be the subnetwork of N induced by $\{u \mid u \leq_N v\}$ and we denote the set of leaf-labels in N_v by $\mathcal{L}(v)$ and abbreviate $\mathcal{L}(N) := \mathcal{L}(\rho_N)$. Such a set is also called a *cluster* of N. Note that, if N is a tree, N_v is the subtree rooted at v. We abbreviate $n := |\mathcal{L}(\rho_N)|$. For any $X \subseteq V(N)$, we let $\mathrm{LCA}_N(X)$ be the set of least common ancestors of X, that is, the minima (wrt. $\leq_N$) among all vertices u of N with $X \leq_N u$ (in particular, if N is a tree, $\mathrm{LCA}_N(X)$ is a single vertex, not a set). If clear from context, we may drop the subscript. Note that, in trees, the LCA of any three vertices has a unique minimum. For any $U \subseteq V(N)$, we denote the result of removing all vertices v that do not have a descendant in

[1] Given a binary network N on the taxa X and some $Y \subseteq X$, Cluster Containment asks if N displays any binary tree T in which $\mathcal{L}(u) = Y$ for any u. This is equivalent to N softly displaying the tree T in which all taxa in $X \setminus Y$ are children of the root and there is another child u of the root with children Y.

U by $N|_L$ and $N||_L$ is the result of *suppressing* all degree-two vertices in $N|_L$. Suppressing a vertex u in N with unique parent p and unique child c refers to the act of removing u and adding the edge pc, unless this edge already exists. Note that, if N is a tree, then $N|_L$ is the smallest subtree of N containing the vertices in L and the root of N and $N||_L$ is the smallest topological minor of N containing the vertices in L and the root of N. A vertex u in N is called *stable* on v if all ρ_N-v-paths contain u. If, for each reticulation u in N there is some leaf ℓ such that u is stable on ℓ, then N is called *reticulation visible*. A network is *binary* if all vertices except the root have degree (=in-degree + out-degree) at most three and the root has degree two. A binary network N_B on three leaves a, b, and c is called a *triplet* and we denote it by $ab|c$ if c is a child of the root of N_B. N_B is called *binary resolution* of a network N if N is a contraction of N_B. In this case, there is a surjective function $\chi : V(N_B) \to V(N)$ such that, contracting all edges uv of N_B with $\chi(u) = \chi(v)$ results in N (more formally, for each $x, y \in V(N)$, the edge xy exists in N if and only if there is an edge between $\chi^{-1}(x)$ and $\chi^{-1}(y)$ in N_B). We call such a function *contraction function* of N_B for N. We suppose that all binary resolutions are minimal, that is, they do not contain biconnected components with exactly one incoming and one outgoing edge. Observe that, when contracting edges of N_B to form N, we never create vertices with in-degree *and* out-degree more than one.

▶ **Observation 1.** *Let N_B be a binary resolution of a network N, let χ be a contraction function of N_B for N, and let $u \in V(N)$. Then, $\chi^{-1}(u)$ does not contain a reticulation and a tree vertex with out-degree more than one.*

If N contains a subgraph S that is isomorphic[2] to a tree T, then we simply say that N contains a subdivision of T. Slightly abusing notation, we consider each vertex $v \in V(T)$ equal to the vertex of S (and, thus, of N) that v is mapped to by an isomorphism. Thus, S consists of $V(T)$ and some vertices of in- and out-degree one. The following definition is paramount.

▶ **Definition 2.** Let N be a network and let T be a tree. Then,

- N *firmly displays* T if and only if N contains a subdivision of T and
- N *softly displays* T if and only if there are binary resolutions N_B of N and T_B of T such that N_B firmly displays T_B.

Definition 2 is motivated by the concept of "hard" and "soft" polytomies (that is, high degree vertices): In phylogenetics, a polytomy is called *firm* or *hard* if it corresponds to a split of multiple species at the same time and *soft* if it represents a set of binary speciations whose order cannot be determined from the available data. In this sense, a polytomy is compatible with another if and only if there is a biological "truth", that is, a binary resolution, that is common to both. Note that, for binary N and T, the two concepts coincide. Furthermore, for trees on the same label-set, the concepts of display and binary resolution coincide.

▶ **Observation 3.** *Let T and T_B be trees on the same leaf-label set and let T_B be binary. Then, T softly displays T_B if and only if T_B is a binary resolution of T.*

Throughout this work we will mostly use the soft variant and we will refer to it simply as "display" for the sake of readability. Note that a binary tree displays another binary tree if and only if they are isomorphic. Thus, in the special case that N is a tree, the "display" relation is symmetrical, leading to the following observation.

[2] In this work, "isomorphic" always refers to isomorphism respecting leaf-labels, that is, all isomorphisms must map a leaf of label λ to a leaf of label λ.

▶ **Observation 4.** *A tree T displays a tree T' if and only if T' displays T.*

Finally, the central problem considered in this work is the following.

Soft Tree Containment
Input: A network N and a tree T
Question: Does N softly display T?

2 Display with Soft Polytomies

The concept of "display" is well-researched for binary trees, in particular, triplets.

▶ **Observation 5** ([4]). *Let T_B be a binary tree and let $a, b, c \in \mathcal{L}(T_B)$. Then, T_B displays $ab|c$ if and only if $\mathrm{LCA}(ab) < \mathrm{LCA}(bc) = \mathrm{LCA}(ac)$. Indeed, T_B is uniquely identified by the set D of displayed triplets, that is, T_B is the only binary tree displaying the triplets in D.*

However, the "display"-relation with soft polytomies lacks a solid mathematical base in the literature. In this section, we develop alternative characterizations of the term "(softly) display". To do this, we use the following characterization of isomorphism for binary trees.

▶ **Observation 6.** *Binary trees T_B and T'_B on the same label-set are isomorphic if and only if, for each $u \in V(T_B)$ and each $Y \subseteq \mathcal{L}(u)$, u has a child v with $\mathcal{L}(v) = Y$ if and only if $\mathrm{LCA}_{T'_B}(\mathcal{L}(u))$ has a child v' with $\mathcal{L}(v') = Y$.*

▶ **Lemma 7.** *Let N and T be trees. Then, N displays T if and only if, for all $u \in V(T)$ and $v \in V(N)$, it holds that $\mathcal{L}(u) \subseteq \mathcal{L}(v)$, $\mathcal{L}(u) \supseteq \mathcal{L}(v)$ or $\mathcal{L}(u) \cap \mathcal{L}(v) = \varnothing$.*

Proof. Since each label appears only once in N and T, it holds that N displays T if and only if there are binary resolutions N^B of N and T^B of T such that N^B and T^B are isomorphic.

"$\Rightarrow$": Let N softly display T. Towards a contradiction, assume that there are $u \in V(N)$ and $w \in V(T)$ such that $\mathcal{L}(u) \not\subseteq \mathcal{L}(v)$, $\mathcal{L}(u) \not\supseteq \mathcal{L}(v)$ and $\mathcal{L}(u) \cap \mathcal{L}(v) \neq \varnothing$, that is, there are $x \in \mathcal{L}(u) \setminus \mathcal{L}(w)$, $y \in \mathcal{L}(u) \cap \mathcal{L}(w)$, and $z \in \mathcal{L}(w) \setminus \mathcal{L}(u)$. Since there are binary resolutions N^B and T^B of N and T, respectively, such that N^B and T^B are isomorphic, there is a vertex u' in N^B with $\mathcal{L}(u') = \mathcal{L}(u)$ and a vertex v' in T with $\mathcal{L}(v') = \mathcal{L}(v)$. Since N^B and T^B are trees and each leaf-label only appears once in each of them, $N^B_{u'}$ contains the leaves x and y but not the leaf z. Analogously, $T^B_{v'}$ contains the leaves y and z but not the leaf x, contradicting N^B being isomorphic to T^B.

"$\Leftarrow$": In order to show the contraposition, suppose that N does not softly display T. Since N does not softly display T, for any binary resolutions N^B of N and T^B of T, it holds that N^B and T^B are not isomorphic. By Observation 6, there are vertices $p \in V(N^B)$ and $q := \mathrm{LCA}_{T^B}(\mathcal{L}(p))$ with children p_1, p_2 and q_1, q_2, respectively, such that $\mathcal{L}(p_1) \neq \mathcal{L}(q_1)$ and $\mathcal{L}(p_1) \neq \mathcal{L}(q_2)$. We will use the fact that $\mathcal{L}(p_1) \uplus \mathcal{L}(p_2) = \mathcal{L}(p) = \mathcal{L}(q) = \mathcal{L}(q_1) \uplus \mathcal{L}(q_2)$.
Case 1: $\mathcal{L}(p_i) \subsetneq \mathcal{L}(q_j)$ for any i, j. Then, there are taxa

$$x \in \mathcal{L}(p_i) \cap \mathcal{L}(q_j) = \mathcal{L}(q_j) \setminus \mathcal{L}(p_{3-i})$$
$$y \in \mathcal{L}(q_j) \setminus \mathcal{L}(p_i) = \mathcal{L}(q_j) \cap \mathcal{L}(p_{3-i}), \text{ and}$$
$$z \in \mathcal{L}(q_{3-j}) = \mathcal{L}(q_{3-j}) \setminus \mathcal{L}(p_i) = \mathcal{L}(p_{3-i}) \setminus \mathcal{L}(q_j).$$

The case where $\mathcal{L}(q_j) \subsetneq \mathcal{L}(p_i)$ holds is analogous.
Case 2: None of $\mathcal{L}(p_1)$, $\mathcal{L}(p_2)$, $\mathcal{L}(q_1)$, and $\mathcal{L}(q_2)$ are subsets of one another. Then, there are taxa x, y, z such that $x \in \mathcal{L}(p_1) \cap \mathcal{L}(q_1)$ $y \in \mathcal{L}(q_1) \setminus \mathcal{L}(p_1)$, and $z \in \mathcal{L}(q_1) \setminus \mathcal{L}(p_1)$. ◀

We can relate the two forms of "display" for triplets in non-binary trees.

▶ **Observation 8.** *Let T be a tree and let $a, b, c \in \mathcal{L}(T)$. Then,*
(a) *T firmly displays $ab|c$ if and only if $\mathrm{LCA}(ab) <_T \{\mathrm{LCA}(ac), \mathrm{LCA}(bc)\}$.*
(b) *T firmly displays $ac|b$ or $bc|a$ if and only if T does not softly display $ab|c$.*

▶ **Lemma 9.** *A tree T on X softly displays a tree T' on $X \Leftrightarrow$ for all $a, b, c \in X$,*

$$T \text{ firmly displays } ab|c \Rightarrow T' \text{ softly displays } ab|c, \text{ and}$$
$$T' \text{ firmly displays } ab|c \Rightarrow T \text{ softly displays } ab|c$$

Proof. "⇒": By Observation 4, it suffices to show the first of the claimed implications, so let $\mathrm{LCA}_T(ab) <_T \mathrm{LCA}_T(abc)$ and assume towards a contradiction that T' does not display $ab|c$. By Observation 8, we can suppose without loss of generality that T' firmly displays $ac|b$. But then, for $u := \mathrm{LCA}_T(ab)$ and $v := \mathrm{LCA}_{T'}(ac)$, we have $a \in \mathcal{L}(u) \cap \mathcal{L}(v)$, $b \in \mathcal{L}(u) \setminus \mathcal{L}(v)$, and $c \in \mathcal{L}(v) \setminus \mathcal{L}(u)$. Thus, by Lemma 7, T does not display T.

"⇐": Towards a contradiction, assume that T does not display T'. By Lemma 7, there are $u \in V(T)$ and $v \in V(T')$ and $a, b, c \in X$ such that $a \in \mathcal{L}(u) \cap \mathcal{L}(v)$, $b \in \mathcal{L}(u) \setminus \mathcal{L}(v)$, and $c \in \mathcal{L}(v) \setminus \mathcal{L}(u)$. Thus, $\mathrm{LCA}_T(ab) <_T \mathrm{LCA}_T(abc)$ and $\mathrm{LCA}_{T'}(ac) <_{T'} \mathrm{LCA}_{T'}(abc)$. By Observation 8, T firmly displays $ab|c$ and T' firmly displays $ac|b$. With the implications of the lemma, we get that T' softly displays $ab|c$ and T softly displays $ac|b$, contradicting Observation 8. ◀

The final ingredient to our alternative characterization is the observation that, in (multi-labeled) trees, edge contraction does not change the ancestor relation.

▶ **Observation 10.** *Let T be a tree, let T' be the result of contracting a vertex u onto its parent v, and let Y and Z be sets of leaves common to T and T'. Then,*
(a) *$\mathrm{LCA}_T(Y) \leq_T \mathrm{LCA}_T(Z) \Leftrightarrow \mathrm{LCA}_{T'}(Y) \leq_{T'} \mathrm{LCA}_{T'}(Z)$ and*
(b) *$\mathrm{LCA}_T(Y) <_T \mathrm{LCA}_T(Z) \Leftarrow \mathrm{LCA}_{T'}(Y) <_{T'} \mathrm{LCA}_{T'}(Z)$.*
We can now prove the following alternative definition of "display".

▶ **Lemma 11.** *Let T be a tree on the label-set X.*
(a) *T displays the leaf-triplet $ab|c$ if and only if $\mathrm{LCA}(ab) \leq \{\mathrm{LCA}(bc), \mathrm{LCA}(ac)\}$.*
(b) *T displays a binary tree T_B on X if and only if T displays all triplets displayed by T_B.*
(c) *T displays a tree T' on X (and vice versa) if and only if there is a binary tree T_B on X displayed by both T and T'.*
(d) *A network N displays T if and only if N contains (as subgraph) a tree T' on X that displays T.*

Proof. (a) By definition, T displays $ab|c$ if and only if there is a binary resolution T_B of T displaying $ab|c$. By Observation 5, T_B displays $ab|c$ if and only if $\mathrm{LCA}_{T_B}(ab) <_{T_B} \mathrm{LCA}_{T_B}(abc) = \mathrm{LCA}_{T_B}(ac) = \mathrm{LCA}_{T_B}(bc)$. Now, since T_B is binary, we cannot have that $\mathrm{LCA}_{T_B}(ab) = \mathrm{LCA}_{T_B}(bc) = \mathrm{LCA}_{T_B}(bc)$ and, thus, $\mathrm{LCA}_{T_B}(ab) \leq_{T_B} \{\mathrm{LCA}_{T_B}(ac), \mathrm{LCA}_{T_B}(bc)\}$ which, by Observation 10, is equivalent to $\mathrm{LCA}_T(ab) \leq_T \{\mathrm{LCA}_T(ac), \mathrm{LCA}_T(bc)\}$.

(b) "⇒": Assume towards a contradiction that a triplet $ab \,|\, c$ of T_B is not displayed by T and recall that $\{\mathrm{LCA}_T(ab), \mathrm{LCA}_T(ac), \mathrm{LCA}_T(bc)\}$ has a unique minimum x. Since, by (a), $\mathrm{LCA}_T(ab) \not\leq_T \mathrm{LCA}_T(abc)$, we have $x <_T \mathrm{LCA}_T(ab) \leq_T \mathrm{LCA}_T(abc)$. Without loss of generality, let $x = \mathrm{LCA}_T(ac)$. Then, by Observation 10, $\mathrm{LCA}_{T_B}(ac) <_{T_B} \mathrm{LCA}_{T_B}(abc)$, implying that T_B displays $ac \,|\, b$. Hence, T_B displays conflicting triples, contradicting Observation 5.

"⇐": Assume towards a contradiction that T does not display T_B. By Lemma 7, there are vertices $u \in V(T)$ and $v_B \in V(T_B)$ such that $\mathcal{L}(u)$ and $\mathcal{L}(v_B)$ intersect, but are not in the

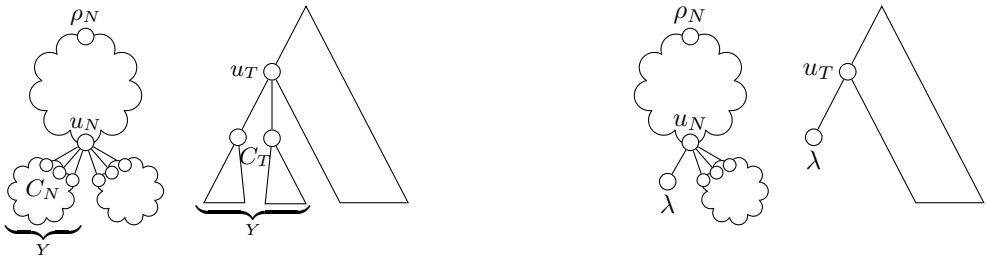


Figure 1 Illustration of Lemma 14: (N, T) left and (N_1, T_1) right.

subset relation, that is, there are $x \in \mathcal{L}(u) \setminus \mathcal{L}(v_B)$, $y \in \mathcal{L}(v_B) \setminus \mathcal{L}(u)$ and $z \in \mathcal{L}(u) \cap \mathcal{L}(v_B)$. Thus, $x, z <_T \mathrm{LCA}_T(xz) \leq_T u <_T \mathrm{LCA}_T(xyz)$ and $y, z <_{T_B} \mathrm{LCA}_{T_B}(yz) \leq_{T_B} v_B <_{T_B} \mathrm{LCA}_{T_B}(xyz)$. Then, by (a), T_B displays $yz \,|\, x$ implying that T displays $yz \,|\, x$ since all triplets displayed by T_B are displayed by T. By (a), we have $\mathrm{LCA}_T(yz) \leq_T \mathrm{LCA}_T(xz)$, implying $x, y, z <_T \mathrm{LCA}_T(xz) \leq_T u$, which contradicts $u <_T \mathrm{LCA}_T(xyz)$.

(c) By definition, T displays T' if and only if there are binary resolutions T_B and T'_B of T and T_B, respectively, such that T_B displays T'_B. Note that, if such trees exist then they are equal since, by (b), T_B displays all triplets displayed by T'_B and, by Observation 5, $T_B = T'_B$. Conversely, by Observation 3, all binary trees on X displayed by T and T' are binary resolutions of T and T'.

(d) We defer this proof to the full version of this paper. ◀

Note that, if N contains a subdivision S of T, then any reticulation in N that is in S has in- and out-degree one in S. Further, contracting an edge between two tree vertices of N cannot break softly displaying T.

▶ **Observation 12.** *Let N be a network that displays a tree T. Then, the result of contracting an edge between two tree-vertices or two reticulations of N displays T.*

Also note that, if N displays T, then the result of removing any label from N displays the result of removing this label from T.

▶ **Observation 13.** *Let N be a network and let T be a tree on X. Then, N displays T if and only if $N|_{X'}$ displays $T|_{X'}$ for each $X' \subseteq X$.*

3 Single-Labeled Trees

In a first step, we suppose that N is a tree. While Lemma 7 already provides the means to solve this case in polynomial time, we aim to be more efficient. If N and T are both binary, this special case is solved using the folklore "cherry reduction": remove a pair of leaves that are siblings in both N and T and label their parents in N and T with the same new label λ. Here, we prove an analog for non-binary trees that allows solving the case that N is a tree in linear time.

▶ **Lemma 14.** *Let N be a network on X with root ρ_N, let T be tree on X, let $u_N \in V(N)$ and $u_T \in V(T)$ and let C_N and C_T be sets of children of u_N and u_T, respectively, such that*
(a) *$\bigcup_{c \in C_N} \mathcal{L}(c) = \bigcup_{c \in C_T} \mathcal{L}(c) =: Y$, and*
(b) *for all $\lambda \in Y$, all ρ_N-λ-paths contain some $c \in C_N$.*
Let $\lambda \in Y$, let $N_1 := N||_{X \setminus (Y - \lambda)}$, let $T_1 := T||_{X \setminus (Y - \lambda)}$, let $N_2 := N||_Y$, and let $T_2 := T||_Y$. Then, N displays T if and only if N_1 displays T_1 and N_2 displays T_2 (see Figure 1).

Proof. Since "⇒" follows directly from Observation 13, we only show "⇐". By Lemma 11, for each $i \in \{1, 2\}$, there is a tree Q_i in N_i (containing the root of N_i) that displays T_i and there is a binary tree T_i^B that is displayed by both Q_i and T_i. We show that the binary tree T_B resulting from replacing the leaf λ in T_1^B by T_2^B is displayed by both T and a subtree Q of N. To this end, note that T is the result of replacing the leaf λ in T_1 by T_2 and let Q be the result of replacing the leaf λ in Q_1 by Q_2. Since T_i^B is displayed by both T_i and Q_i for all $i \in \{1, 2\}$, the following argument holds for both T and Q, but we only state it for T. To show that T displays T_B, it suffices to prove that T displays all triplets displayed by T_B (by Lemma 11(b)). Towards a contradiction, assume that T_B displays a triplet $xy|z$ that T does not display.

Case 1: $x, y \in Y$. If z is also in Y, then $xy|z$ is displayed by T_2^B and, thus, by T_2 and by T. If $z \notin Y$, then $\mathrm{LCA}_T(xy) \leq_T \mathrm{LCA}_T(Y) \leq_T u_T \leq_T \{\mathrm{LCA}_T(xz), \mathrm{LCA}_T(yz)\}$ by (a) (and (b) when arguing for Q instead of T) and, by Lemma 11(a), T displays $xy|z$.

Case 2: x or y is not in Y. Without loss of generality, let $x \notin Y$. If also $y \notin Y$, then λ can take the role of z in the assumption, that is, T_B displays $xy|\lambda$ but T does not. But then, T_B^1 displays $xy|\lambda$ but T_1 does not, contradicting the fact that T_1 displays T_B^1. Thus, $y \in Y$ and, completely analogously, $z \in Y$. But then, $\mathrm{LCA}_{T_B}(yz) \leq_{T_B} \mathrm{LCA}_{T_B}(Y) < \mathrm{LCA}_{T_B}(xy)$ which, by Lemma 11(a), contradicts T_B displaying $xy|z$.

Finally, let T^* be the result of contracting $\mathrm{LCA}_Q(Y)$ (that is, the former root of T_2^*) onto its parent in Q. Then, T^* is a subtree of N since N is (isomorphic to) the result of replacing ℓ by N_2 in N_1 and contracting the the root of N_2 onto its parent in the result. Since Q displays T_B, so does T^* (by Observation 12). Thus, T^* is a subtree of N that displays T and, by Lemma 11(d) N displays T. ◀

In the following, the operation of *splitting off* a subnetwork B with root u in a network N means to
remove B and
add a new leaf labeled $\lambda \notin X$ to u.
This gives rise to the networks N_1 (containing the new leaf λ) and $N_2 := B$. Lemma 14 implies correctness of the following reduction rule.

▶ **Reduction Rule 1.** *Let (N, T) be an instance of* Soft Tree Containment*, let B be a lowest biconnected component (such that B does not consist of a leaf and a non-leaf) or a cherry of N with root u. Then, split off B from N and split-off $T_{\mathrm{LCA}_T(\mathcal{L}(u))}$ from T (giving the new leaf in N and T the same new label λ).*

Note that Reduction Rule 1 can be applied exhaustively in linear time. This is because
(a) biconnected components can be found in linear time [12], and
(b) no biconnected component of N (except B) is modified by application of Reduction Rule 1.
Now, if N is a (single-labeled) tree, then Reduction Rule 1 splits-off only cherries from N and each such cherry can be checked against the subtree split-off from T in linear time.

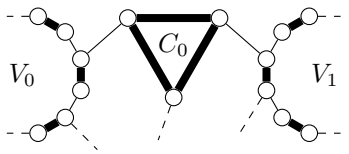
▶ **Theorem 15.** Soft Tree Containment *can be solved in linear time if N and T are trees.*

4 Tree Containment in Multilabeled Trees

To show that Soft Tree Containment is NP-hard even when restricting N to be a multilabeled tree, we reduce from 2-Union Independent Set, which asks if a graph $(V, E_1 \cup E_2)$ has a size-k independent set, and which is NP-hard even if (V, E_1) is a collection

of disjoint K_2s (that is, a matching) and (V, E_2) is a collection of disjoint P_2s and P_3s [16]. For our reduction, we allow (V, E_1) to also contain K_3s and demand that k equals the number of cliques in (V, E_1). To prove that this variant remains NP-hard, we slightly modify the reduction from 3-SAT given by van Bevern et al. [16].

▶ **Construction 1.** *Consider an instance φ with n variables x_i and m clauses c_j of* 3-SAT *such that each variable occurs at least twice in φ and at most once in each clause. For each variable x_i, let J_i be the list of indices of clauses that contain x_i or $\neg x_i$ and let $J_i[\ell]$ denote the ℓ^{th} element of this list. Construct a graph (V, E) as follows. For each variable x_i, construct a cycle V_i of $2|J_i|$ vertices: $(u_i^1, \overline{u}_i^1, u_i^2, \overline{u}_i^2, \ldots)$. For each clause c_j on the variables x_i, x_k, x_ℓ, construct a triangle $C_j = (w_j^i, w_j^k, w_j^\ell)$. For each variable x_i and each $\ell \leq |J_i|$, connect $w_{J_i[\ell]}^i$ to $\overline{u}_i^\ell$ if $c_{J_i[\ell]}$ contains x_i, and to u_i^ℓ if $c_{J_i[\ell]}$ contains $\neg x_i$. Now, (V, E_1) (bold in the figure) consist of all triangles and all edges $\{u_i^j, \overline{u}_i^{j+1 \bmod |J_i|}\}$ while E_2 contains all other edges.*



Note that (V, E_1) consists of disjoint K_2s and K_3s and (V, E_2) consist exclusively of P_3s. Also note that this generalizes to k-SAT but (V, E_1) becomes a collection of disjoint K_2s and K_ks.

▶ **Lemma 16.** *φ is satisfiable if and only if (V, E) has a size-k independent set, where k is the number of cliques in (V, E_1).*

Proof. Note that
k equals the number of cliques in (V, E_1),
each clique contains at most one independent vertex, and
all vertices in V are incident with some edge in E_1.
Hence, (V, E) contains a size-k independent set, if and only if a largest independent set in (V, E) contains exactly one vertex of each clique in (V, E_1). We will first show that if (V, E) contains an independent set of size k, then φ is satisfiable and afterwards the other direction.

"$\Leftarrow$": Let I be an independent set of size k in (V, E). Then, for each i, I contains either u_i^1 or $\overline{u}_i^1$. By construction of V_i, it holds that if $u_i^h \in I$ for some h, then $u_i^\ell, v_i^\ell \in I$ for all $\ell \leq |J_i|$. Analogously, if $\overline{u}_i^h \in I$ for some h, then $\overline{u}_i^\ell, \overline{v}_i^\ell \in I$ for all $\ell \leq |J_i|$. Consider any vertex w_j^i in the clause gadgets that is in I. Then, w_j^i has a unique neighbor in the variable gadget of x_j which is either u_j^h for some h if $\neg x_j$ occurs in clause i or $\overline{u}_j^\ell$ otherwise. If the neighbor is u_j^h, then all vertices $\overline{u}_j^\ell$ with $1 \leq \ell \leq |J_j|$ are in I and otherwise all vertices u_j^ℓ.

We set x_i to true if u_i^1 is in I and to false if $\overline{u}_i^1$ is in I. Consider any clause c_j in φ. The literal whose corresponding vertex is in I is then set to true as its neighboring vertex u is not in I and u has a neighbor u_i^h for some h if x_i occurs in c_j and a neighbor $\overline{u}_i^h$ for some h if $\neg x_i$ appears in c_j. Since each clause has at least one variable set to true, φ is satisfiable.

"$\Rightarrow$": We will now show that if φ is satisfiable, then (V, E) contains an independent set of size k. Let β be a satisfying assignment for φ. We construct an independent set I for (V, E) as follows. For each x_i and each $\ell \leq |J_i|$, the set I contains the vertices u_i^ℓ and v_i^ℓ if $\beta(x_i) = 1$, and the vertices $\overline{u}_i^\ell$ and $\overline{v}_i^\ell$, otherwise. For each clause c_j we pick one literal that is satisfied by our assignment of the variables and put the corresponding vertex into I. Observe that I is of size k as exactly one vertex of each clique in (V, E_1) is in I. Further, I is independent since, in each variable gadget, we pick every second vertex and, if a vertex

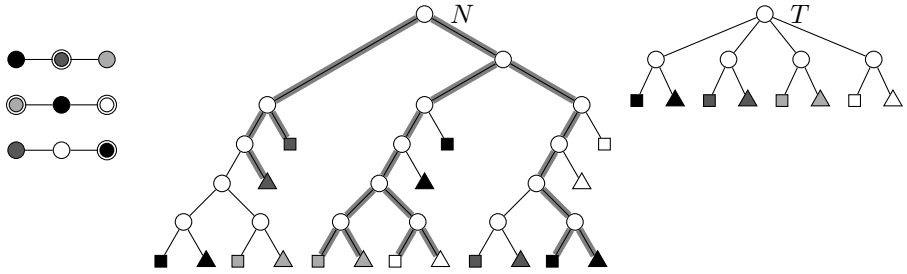
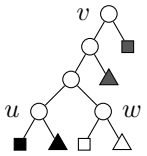


Figure 2 Illustration of Construction 2. **Left:** the initial instance of 2-Union Independent Set with 4 colors (●,●,●,○) and a size-4 solution encircled. **Right:** the non-binary tree T (boxes and triangles indicating label i_1 and i_2 for a color i). **Middle:** the binary multi-labeled tree N with a subdivision of T (bold, gray) corresponding to the solution to the left instance.

in a clause gadget is picked, then its neighbor in the corresponding variable gadget is not picked. ◀

We reduce this version of 2-Union Independent Set to Soft Tree Containment for multilabeled trees. To this end, we use an equivalent formulation where each clique in (V, E_1) is represented by a color. The problem then becomes the following: Given a vertex-colored collection of P_3s, select exactly one vertex per color such that all selected vertices are independent. Note that the number of occurrences of each color equals the size of its corresponding clique in (V, E_1).

▶ **Construction 2** (See Figure 2). *Given a vertex-colored collection G of P_3s constructed by Construction 1, we construct a multi-labeled tree N and a tree T as follows. Construct T by first creating a star that has exactly one leaf of each color occurring in G and then, for each leaf x with color i, adding two new leaves colored i_1 and i_2, respectively, and removing the color from x. Construct N from G as follows: For each P_3 (u, v, w) where black, gray, and white denote the colors of u, v, and w, respectively, construct the binary tree depicted below, where a box or a triangle colored i represents color i_1 or i_2, respectively. Then, add any binary tree on $|V(G)|$ leaves and identify its leaves with the roots of the constructed subtrees. Notice $u, v, w \in V(G) \cap V(N)$.*



▶ **Lemma 17.** *Construction 2 is correct, that is, N displays T if and only if the given collection G of P_3s has a colorful independent set using each color exactly once.*

Proof. Note that N is binary and let k be the number of colors in G.

"⇒": Let N display T, that is, N contains a binary tree S displaying T which, by Lemma 11 is equivalent to T displaying S. Consider any color i occurring in G. Then, S contains leaves u_1 and u_2 in S labeled i_1 and i_2, respectively, and we denote their least common ancestor in S by u^i. If u_1 and u_2 are neither siblings, nor in an uncle-nephew-relation[3], then we modify S to include the sibling/uncle of u_1 in N into S instead of u_2.

[3] Two vertices are in an *uncle-nephew relation* if the sibling of one is the parent of the other

Thus, we do not lose generality by assuming that u_1 and u_2 are either siblings or in an uncle-nephew-relation. We show that the set $Q = \bigcup_i u^i$ is a size-k colorful independent set in G. First, for each color i, we know that S contains exactly one leaf labeled i_1 and one leaf labeled i_2, so u^i is unique and, by construction of N, no two u^i coincide, implying that Q contains exactly one vertex of each color. Towards a contradiction, suppose that Q is not independent in G, that is, there are colors i and j such that u^i and u^j are adjacent in G. Without loss of generality, u^i is the center of a P_3 in G, implying that S contains the subtree $((((j_1, j_2), i_1), i_2)$ (that is, a caterpillar with leaves labeled j_1, j_2, i_1, i_2 in preorder). But then, $j_1 i_1 | i_2$ is displayed by S but not by T, thereby contradicting Definition 2(b).

"$\Leftarrow$": Let Q be a size-k colorful independent set of G, let L be the set of leaves that, for each $u \in Q$ of color i, contains the leaves labeled i_1 and i_2 in N_u, and let $S := N|_L$. Note that S is a subgraph of N and, as N is binary, S is a subdivision of a binary tree. Since Q contains exactly one vertex of each color in G, we know that S contains all labels that occur in T. By Definition 2(d), to show that N displays T, it suffices to show that S displays T. To this end, assume that S displays a triplet $xy|z$ that T does not display. Then Definition 2(a) lets us assume $\mathrm{LCA}_T(xz) <_T \{\mathrm{LCA}_T(xy), \mathrm{LCA}_T(yz)\}$ without loss of generality. Thus, $x = i_1$, $z = i_2$, and $y = j_1$ for colors $i \neq j$. By Definition 2(a), we have $\mathrm{LCA}_S(i_1 j_1) \leq_S \mathrm{LCA}_S(i_1 i_2)$. Then, i_1 and i_2 cannot form a cherry in S and, thus, $S|_{\{i_1,i_2,j_1,j_2\}}$ is the subtree $(((j_1, j_2), i_1), i_2)$. By construction of S, this implies that Q contains two vertices of a P_3 in G, one of color i and one of color j, and the latter is in the middle, contradicting independence of Q in G. ◀

▶ **Theorem 18.** Soft Tree Containment *is NP-hard, even if N is a binary 3-labeled tree.*

Note that the number of occurrences of each label in N equals the number of occurrences of each color in G which, in turn, equals the size of a largest clique in (V, E_1) (instance of 2-Union Independent Set), which equals the size of a largest clause (instance of 3-SAT), we can state the following generalization of Theorem 18.

▶ **Corollary 19.** *For each k, k-SAT reduces to* Soft Tree Containment *on binary k-labeled trees. Further, CNF-SAT reduces to* Soft Tree Containment *on binary multilabeled trees.*

Corollary 19 immediately raises the question of what happens in the case that N is a 2-labeled tree and we address this question in Section 4.1. Note that, for Soft Tree Containment, the case that N is a multilabeled tree reduces straightforwardly to the case that N is a reticulation-visible network, simply by merging all leaves with the same label i into one reticulation and adding a new child labeled i to it.

▶ **Corollary 20.** Soft Tree Containment *is NP-hard on reticulation-visible networks, even if the maximum in-degree is three and the maximum out-degree is two.*

Theorem 18 and Corollary 20 stand in contrast with results for (Strong) Tree Containment, which is linear-time solvable in both cases [18, 7].

4.1 2-Labeled Trees

In the following, N is a 2-labeled tree and T is a (single-labeled) tree. To solve Soft Tree Containment in this case, we compute a mapping $M : V(T) \to 2^{V(N)}$ such that $M(u)$ contains the at most two minima (with respect to $\leq_N$) among all vertices v of N such that N_v displays T_u. If N displays T, there is a single-labeled subtree S of N that displays T. If, for each $u \in V(T)$, we have $\mathrm{LCA}_S(\mathcal{L}(u)) \in M(u)$, then we call S *canonical* for T. We show that such a canonical subtree always exists.

▶ **Lemma 21.** *N displays T if and only if N has a canonical subtree for T.*

Proof. As "⇐" is evident, we just prove "⇒". To this end, let S be a single-labeled subtree of N that is a subdivision of T. If S is not canonical, then there is some $u \in V(T)$ with $x := \mathrm{LCA}_S(\mathcal{L}(u)) \notin M(u)$. Since S_x displays T_u, so does N_x. Thus, by definition of M, there is some $y \in M(u)$ with $y <_N x$ (recall that $x \notin M(u)$). But then, we can replace the subtree of S rooted at x with the unique x-y-path in N and the subtree of N_y displaying T_u. Iterating this construction yields a canonical subtree of N for T. ◀

To compute M, we consider vertices $u \in V(T)$ and $\rho \in V(N)$ in a bottom-up manner and check if N_ρ displays T_u. For each $v \in V(T_u)$ with parent p in T_u, each $x \in M(v)$ has at most one ancestor y in $M(p)$ since M contains only minima. For $v = u$, we let $y := \rho$. In both cases, we call the unique x-y-path in N_ρ the *ascending path* of x. A crucial lemma about ascending paths is the following.

▶ **Lemma 22.** *Let S be a canonical subtree of some N' for some T' and let $u, v \in V(T')$ not be siblings. Let $\mathrm{LCA}_S(\mathcal{L}(u))$ and $\mathrm{LCA}_S(\mathcal{L}(v))$ have ascending paths r and q, respectively. Then, r and q are edge-disjoint.*

Proof. Note that, if $u <_{T'} v$ then $\mathrm{LCA}_S(\mathcal{L}(p)) \leq_S \mathrm{LCA}_S(\mathcal{L}(v))$ where p is the parent of u in T'. Thus, the highest vertex of r (with respect to $\leq_{N_\rho}$) is a descendant of the lowest vertex of q and, hence, the lemma holds. Thus, we suppose in the following that u and v are incomparable in T'.

Towards a contradiction, assume that there is a vertex $z \in V(S)$ that is internal vertex of both r and q and, hence, is an ancestor of both u and v in T'. Then, $\mathcal{L}(u) \uplus \mathcal{L}(v) \subseteq \mathcal{L}(z)$. Further, since u and v are not siblings, one of u and v has a parent $p <_{T'} \mathrm{LCA}_{T'}(uv)$. Without loss of generality, let p be the parent of u, implying $\mathcal{L}(p) \cap \mathcal{L}(z) \supseteq \mathcal{L}(u) \neq \varnothing$ and $\mathcal{L}(z) \setminus \mathcal{L}(p) \supseteq \mathcal{L}(v) \neq \varnothing$. Since S is canonical, we have $\mathrm{LCA}_S(\mathcal{L}(p)) \in M(p)$ and, thus, the ascending path r of u ends in $\mathrm{LCA}_S(\mathcal{L}(p))$. Hence, as z is an internal vertex of r, it holds that $z <_S \mathrm{LCA}_S(\mathcal{L}(p))$, implying $\mathcal{L}(p) \setminus \mathcal{L}(z) \neq \varnothing$. Since S displays T', the three established relations between $\mathcal{L}(p)$ and $\mathcal{L}(z)$ contradict Lemma 7. ◀

Clearly, N displays T if and only if $M(\rho_T) \neq \varnothing$, where ρ_T is the root of T. Further, computation of $M(u)$ is trivial if u is a leaf. Thus, in the following, we show how to compute $M(u)$ given $M(v)$ for all $v \in V(T_u) - u$.

In a first step, compute $N|_L$ where L is the set of leaves of N whose label occurs in T_u. Then, we know that $M(v) \subseteq V(N|_L)$ for all $v \in V(T_u)$. Second, we mark all vertices ρ in $N|_L$ such that, for each child u_i of u in T, there is some $x_i \in M(u_i)$ with $x_i \leq_{N|_L} \rho$. For each marked vertex ρ in a bottom-up manner, we test whether N_ρ displays T_u using the following formulation as a 2-SAT problem[4].

▶ **Construction 3.** *Construct $\varphi_{u\to\rho}$ as follows. For each $v \in V(T_u) - u$,*

- **(i)** *for each $y \in M(v)$, introduce a variable $x_{v\to y}$.*
- **(ii)** *add the clause $\bigoplus_{y\in M(v)} x_{v\to y}$ (recall that $|M(v)| \leq 2$).*
- **(iii)** *if the parent p of v in T_u is not u then, for all $y \in M(v)$ and all $z \in M(p)$ with $y \not\leq_N z$, add the clause $x_{v\to y} \Rightarrow \neg x_{w\to z}$.*
- **(iv)** *for each $w \in V(T_u) - u$ that is not a sibling of v and each $y \in M(v)$ and each $z \in M(w)$ such that the ascending paths of y and z share an edge, add the clause $x_{v\to y} \Rightarrow \neg x_{w\to z}$.*

[4] We are using the XOR operation ($(x \oplus y) := (x \vee y) \wedge (\neg x \vee \neg y)$) as well as implications ($(x \Rightarrow y) := (\neg x \vee y)$) in the construction, which can be formulated as clauses with two variables as shown.

By definition of $M(u)$, no two vertices in $M(u)$ can be in an ancestor-descendant relation. Thus, we can ignore all ancestors of a vertex ρ that satisfies $\varphi_{u\to\rho}$ and we can assume that no strict ancestor of our current ρ satisfies $\varphi_{u\to z}$.

▶ **Lemma 23.** *$\varphi_{u\to\rho}$ is satisfiable if and only if N_ρ displays T_u.*

Proof. "⇐": Let S be a canonical subtree of N_ρ for T_u and let β be an assignment for $\varphi_{u\to\rho}$ that sets each $x_{v\to y}$ to 1 if and only if $y = \mathrm{LCA}_S(\mathcal{L}(v))$. Since the LCA of $\mathcal{L}(v)$ in S is unique, all clauses of type (ii) are satisfied by β. If a clause of type (iii) is not satisfied, then there is some v with parent p in T_u such that $y \leq_N z$ for some $y \in M(v)$ and $z \in M(p)$ and $\beta(x_{v\to y}) = 1$ and $\beta(x_{p\to z}) = 0$. Let $z' \in M(p) - z$ with $\beta(x_{p\to z'}) = 1$, which exists since all clauses of type (ii) are satisfied. Since $\mathcal{L}(p) \supseteq \mathcal{L}(v)$, we know that $y \leq_S z'$ and, as S is a subtree of N, we have $y \leq_N z'$, implying $z \leq_N z'$ or $z' \leq_N z$, which contradicts the construction of M. If a clause of type (iv) is not satisfied, then there are $x_{v\to y}$ and $x_{w\to z}$ such that v and w are not siblings in T, $\beta(x_{v\to y}) = \beta(x_{w\to z}) = 1$, and the ascending paths of $y = \mathrm{LCA}_S(\mathcal{L}(v))$ and $z = \mathrm{LCA}_S(\mathcal{L}(w))$ share an edge. But this contradicts Lemma 22.

"⇒": Let β be a satisfying assignment for $\varphi_{u\to\rho}$. Let $\psi \subseteq V(T) \times V(N)$ be a relation such that $(v, y) \in \psi$ if and only if $\beta(x_{v\to y}) = 1$. Since β satisfies the clauses of type (ii), ψ describes a function and, slightly abusing notation, we call this function ψ. Let Y be the image of ψ and let $S := N|_{Y\cup\{\rho\}}$. Note that, for all $v <_T u$ with parent $p \neq u$, we know that $\psi(v) \leq_N \psi(p)$, since β satisfies the clauses of type (iii). Thus, for all $v, w \in V(T_u) - u$, we have $w \leq_T v \Rightarrow \psi(w) \leq_N \psi(v) \Rightarrow \psi(w) \leq_S \psi(v)$ We show for all $(v, y) \in \psi \cup \{(u, \rho)\}$ that $y = \mathrm{LCA}_S(\mathcal{L}(v))$ and S_y is a canonical subtree of N_y for T_v. The proof is by induction on the height of v in T. Clearly, if v is a leaf, y is a leaf with the same label and the claim follows. Otherwise, suppose that the claim holds for all $w <_T v$. Towards a contradiction, assume that S_y does not display T_v. By Lemma 7, there are $w \in V(T_v)$ and $z \in V(S_y)$ such that there are leaves $a \in \mathcal{L}(z) \setminus \mathcal{L}(w)$, $b \in \mathcal{L}(w) \setminus \mathcal{L}(z)$, and $c \in \mathcal{L}(w) \cap \mathcal{L}(z)$. Note that $\mathrm{LCA}_T(bc) \leq_T w <_T \{\mathrm{LCA}_T(ab), \mathrm{LCA}_T(ac)\}$. Let α be the highest ancestor of a in T with $b \not\leq_T \alpha$ and let p_α be its parent in T. Let γ be the highest ancestor of c in T with $b \not\leq_T \gamma$ and let p_γ be its parent in T. Since $b, c <_T w$ and $a \not\leq_T w$, we know that $p_\gamma <_T p_\alpha$, implying that α and γ are not siblings in T. Then, as $\mathrm{LCA}_S(ac) \leq_S z <_S \{\mathrm{LCA}_S(ab), \mathrm{LCA}_S(bc)\}$, $\mathrm{LCA}_S(ab) \leq_S \psi(p_\alpha)$, and $\mathrm{LCA}_S(bc) \leq_S \psi(p_\gamma)$, we know that the ascending paths of $\psi(\alpha)$ and $\psi(\gamma)$ share an edge, contradicting (iv). ◀

▶ **Theorem 24.** Soft Tree Containment *can be solved in $O(n^3)$ time on instances (N, T) for which N is a 2-labeled tree.*

Proof. As correctness follows from Lemma 23, we only show the running time. To this end, note the $N|_L$ can be computed in $O(|L|) = O(|\mathcal{L}(u)|)$ time (see, for example [3, Section 8]). To mark all vertices of $N|_L$ that, for each child u_i of u in T, have an ancestor in $M(u_i)$, we compute the restriction of $N|_L$ to $\bigcup_i M(u_i)$. Again, this can be done in $O(\deg_T(u))$ time. For each vertex in this restriction, we can store the set of leaves that descend from it. In a bottom-up manner, we can thus mark the correct vertices in $O(\deg_T(u)^2)$ time.

We construct $\varphi_{u\to\rho}$ for each pair (u, ρ) as follows. To check $y \not\leq_N z$ efficiently in Construction 3(iii), we can prepare a 0/1-matrix with an entry for each pair of vertices in N. This table has size $O(n^2)$ and can be computed in the same time by a simple bottom-up scan of N. To construct the clauses of type (iv), we first order the vertices in N_ρ. For each v in this order, we construct its ascending path in $O(|N_\rho|)$ time and store v in all edges on this path. Thus, when constructing the clauses of type (iv) for a vertex v, we can merge the lists of vertices whose ascending path shares an edges with that of v. Thus, $\varphi_{u\to\rho}$ can be constructed and solved in $O(|N_\rho|^2) = O(|\mathcal{L}(u)|^2)$ time and the total time to decide whether N displays T is $O(\sum_{u\in V(T)} |\mathcal{L}(u)|^2) = O(n^3)$. ◀

Theorem 24 implies [5] that we can solve bifurcating reticulation-visible networks in polynomial time, complementing Corollary 20.

▶ **Corollary 25.** Soft Tree Containment *can be solved in* $O(n^3)$ *time on reticulation-visible networks of in-degree at most two.*

5 Conclusion

We introduced a practically relevant variant of the Tree Containment problem handling soft polytomies and showed that its (classical) complexity depends heavily on the maximum in-degree in the network. Multiple avenues are opened for future work. Motivated by our hardness result, the search for parameterized or approximative algorithms is a logical next step. Previous work for Tree Containment [8, 18] might lend promising ideas and parameterizations to this effort. While multi-labeled trees were our starting point to analyze Soft Tree Containment, only the hardness result (Theorem 20) is transferable to multi-labeled networks, leaving many open questions in this direction. Finally, given the close relationship to Cluster Containment, (see Section 1), we hope to apply ideas and methods used there to also attack Soft Tree Containment. In particular, we hope that the ideas in Theorem 24 can be adapted since Cluster Containment seems to exhibit a close relationship to SAT [9]—similar to what we exploited to prove Theorem 24.

References

1 Magnus Bordewich and Charles Semple. Reticulation-visible networks. *Advances in Applied Mathematics*, 78:114–141, 2016.

2 Joseph Minhow Chan, Gunnar Carlsson, and Raul Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

3 Richard Cole, Martin Farach-Colton, Ramesh Hariharan, Teresa Przytycka, and Mikkel Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

4 A Dress, Katharina Huber, J Koolen, Vincent Moulton, and A Spillner. *Basic Phylogenetic Combinatorics*. Cambridge University Press, 2004.

5 Jittat Fakcharoenphol, Tanee Kumpijit, and Attakorn Putwattana. A faster algorithm for the tree containment problem for binary nearly stable phylogenetic networks. In *12th International Joint Conference on Computer Science and Software Engineering (JCSSE'15)*, pages 337–342. IEEE, 2015.

6 Philippe Gambette, Andreas D. M. Gunawan, Anthony Labarre, Stéphane Vialette, and Louxin Zhang. Locating a tree in a phylogenetic network in quadratic time. In *Proceedings of the 19th Annual International Conference on Research in Computational Molecular Biology (RECOMB'15)*, volume 9029 of *LNCS*, pages 96–107. Springer, 2015.

7 Andreas D. M. Gunawan. Solving tree containment problem for reticulation-visible networks with optimal running time. *CoRR*, abs/1702.04088, 2017.

8 Andreas D. M. Gunawan, Bingxin Lu, and Louxin Zhang. A program for verification of phylogenetic network models. *Bioinformatics*, 32(17):i503–i510, 2016.

9 Andreas D. M. Gunawan, Bingxin Lu, and Louxin Zhang. Fast methods for solving the cluster containment problem for phylogenetic networks. *CoRR*, 1801.04498, 2018.

[5] See [18] for the corresponding reduction.

10 Andreas D.M. Gunawan, Bhaskar DasGupta, and Louxin Zhang. A decomposition theorem and two algorithms for reticulation-visible networks. *Information and Computation*, 252:161–175, 2017.

11 Dan Gusfield. *ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks*. MIT Press, 2014.

12 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

13 Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

14 Iyad A Kanj, Luay Nakhleh, Cuong Than, and Ge Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401(1-3):153–164, 2008.

15 Todd J Treangen and Eduardo PC Rocha. Horizontal transfer, not duplication, drives the expansion of protein families in prokaryotes. *PLoS Genet*, 7(1):e1001284, 2011.

16 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *J. Scheduling*, 18(5):449–469, 2015. doi:10.1007/s10951-014-0398-5.

17 Leo Van Iersel, Charles Semple, and Mike Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110(23):1037–1043, 2010.

18 Mathias Weller. Linear-time tree containment in phylogenetic networks. *CoRR*, 1702.06364, 2017.

On the Size of Outer-String Representations

Therese Biedl[1]
Cheriton School of Computer Science, University of Waterloo
Waterloo, Canada
biedl@uwaterloo.ca

Ahmad Biniaz[2]
Cheriton School of Computer Science, University of Waterloo
Waterloo, Canada
ahmad.biniaz@gmail.com

Martin Derka[3]
School of Computer Science, Carleton University
Ottawa, Canada
mderka@uwaterloo.ca

Abstract

Outer-string graphs, i.e., graphs that can be represented as intersection of curves in 2D, all of which end in the outer-face, have recently received much interest, especially since it was shown that the independent set problem can be solved efficiently in such graphs. However, the runtime for the independent set problem depends on N, the number of segments in an outer-string representation, rather than the number n of vertices of the graph. In this paper, we argue that for some outer-string graphs, N must be exponential in n. We also study some special string graphs, viz. monotone string graphs, and argue that for them N can be assumed to be polynomial in n. Finally we give an algorithm for independent set in so-called strip-grounded monotone outer-string graphs that is polynomial in n.

2012 ACM Subject Classification Theory of computation → Computational geometry, Mathematics of computing → Graph theory

Keywords and phrases string graph, outer-string graph, size of representation, independent set

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.10

1 Introduction

A *string graph* is a graph $G = (V, E)$ that has a *string representation*, i.e., an assignment of curves in the plane to the vertices in such a way that two vertices v, w are connected by an edge (v, w) if and only if their corresponding curves $\mathbf{v}, \mathbf{w}$ intersect. In this paper, we only consider string representations where any two curves $\mathbf{v}$ and $\mathbf{w}$ intersect in a finite set of points (denoted $\mathbf{v} \cap \mathbf{w}$). We will always use bold-face $\mathbf{v}$ to denote the curve of a vertex v.

The study of string graphs goes back over 50 years, see e.g. [24, 7]. It is known that every planar graph is a string graph [7], but in general, testing whether a graph is a string graph is NP-complete [15, 20, 22]. Many variants of string graphs have been studied in the literature. Of chief interest to us are the so-called *outer-string* graphs, which have a string representation such that for every vertex v the curve $\mathbf{v}$ has at least one endpoint on

[1] Supported by NSERC.
[2] Supported by NSERC Postdoctoral Fellowship.
[3] Supported by NSERC Vanier fellowship while author was a student at University of Waterloo.

the outer-face of the string representation. See some recent articles [2, 3] for some results concerning outer-string graphs and some subclasses.

The class of outer-string graphs includes the circle graphs (i.e., graphs of intersections of chords of a circle), so any decision problem that is NP-hard for circle-graphs is also NP-hard for outer-string graphs. This includes, among others, the Coloring problem and the Hamiltonian Cycle problem [12, 5]. However, it does not include the maximum independent set problem, i.e., the problem where we are given a graph with vertex-weights (not necessarily uniform), and we want to find the maximum-weight vertex-set I such that no two vertices in I are adjacent.

The work in the current paper was inspired by a result from 2015 in which Keil, Mitchell, Pradhan and Vatshelle presented a poly-time algorithm for maximum independent set in outer-string graphs [14]. They assume that an outer-string representation R is given, and "poly-time" means polynomial in the size of R (typically measured by assuming that R uses only polygonal lines and counting the number of segments). Their algorithm runs in time $O(N^3)$ where N is the size of R.

Since the algorithm of Keil et al. [14] requires the representation to be given, the following question remains open: Given an outer-string graph G (but no outer-string representation), can we find a maximum independent set of G in polynomial time? One natural approach to this would be to try to find an outer-string representation of G. There are two obstacles here though. First, no algorithm is known to find such a representation (but this problem is also not known to be NP-hard). Second, even if such an algorithm were known, what would be the size N of the resulting outer-string representation? There are string graphs for which any string representation requires exponential size [16]. What can be said about the size of a representation required for outer-string graphs? This is the main topic of this paper.

1.1 Related results

We provide here an overview of some algorithmic results on string graphs. Since planar graphs are string graphs [7], all problems that are NP-hard for planar graphs remain NP-hard for string graphs. The converse statement is not true because there are problems that are polynomial for planar graphs (e.g. maximum clique) but NP-hard for string graphs [19].

String representations have been used to obtain better approximation algorithms, especially for independent set. Matoušek [18] showed that every string graph with m edges admits a vertex separator (a set S such that all components of $G - S$ have at most $\frac{2}{3}n$ vertices) of size $O(\sqrt{m}\log m)$. Fox and Pach conjectured that every string graph has a separator of size $O(\sqrt{m})$ [9]. This was proved, first for *k-intersecting string graphs* (any two strings intersect at most k times) [8] and very recently for all string graphs [17]. One example of a result based on separators is an n^{ε}-approximation algorithm for maximum independent set in k-intersecting string graphs by Fox and Pach [10]. Har-Peled and Quanrund [13] showed that separator theorems are applicable for approximation algorithms for all sparse string graphs. However, none of these results seems to lead to approximation algorithms with factors better than $O(n^{\varepsilon})$ for all string graphs.

A *segment graph* is a string graph that has a string representation in which all strings are line segments. Agarwal and Mustafa [1] proved that if all these segments intersect a given line, then an independent set of size $\sqrt{\alpha}$ can be computed in $O(n^3)$ time where α is the size of a maximum independent set. They also showed that any segment graph can be split into $O(\log n)$ subgraphs that are segment graphs for which all segments intersect one line. They used this to obtain an independent set of size $\sqrt{\alpha/\log(n/\alpha)}$ for all segment graphs.

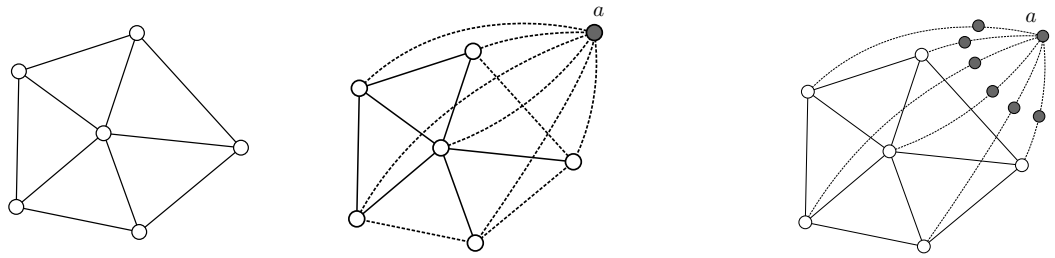


Figure 1 A graph G, its apex graph, and the subdivided apex graph G^+.

1.2 Our contribution

In this paper we show that for some n-vertex outer-string graphs any outer-string representation requires $\Omega(2^{n/10})$ crossings, and consequently exponentially many segments. This result implies that the independent set algorithm of Keil et al. [14] does not run in polynomial time for all outer-string graphs, but only for those that have a polynomial-size outer-string representation. Our result also motivates exploration of algorithms whose running times have lower dependency on the size of representation.

We next explore graph classes that do have small string representations. We consider a natural subclass of string graphs, the *monotone string graphs*, where every string is a y-monotone curve, and argue that any such representation can be transformed into one of polynomial size. Combining this with Keil et al.'s algorithm implies that for monotone outer-string graphs, the maximum independent set is polynomial in n, however, the running time is rather large. We also study a special case where every monotone string has one endpoint on an enclosing strip. For this case we present a dynamic programming algorithm that finds a maximum independent set in $O(n^6)$ time and a 2-approximation in $O(n^3)$ time.

1.3 Outer-string graphs and apices

For our proof that some outer-string graphs require large representations, it will help to have a characterization of outer-string graphs. Although this characterization is simple (a similar approach has been used by Middendorf and Pfeiffer to characterize so-called cylinder graphs [20]), to our knowledge it has not been given before. Let G be a graph. The *apex graph* H of G is the graph obtained from G by adding a new vertex a connected to all vertices in G. The *subdivided apex graph* of G, denoted by G^+, is obtained from H by subdividing every edge incident to a. See Figure 1. One can easily show the following (see [6] for details):

▶ **Lemma 1.** *Graph G is an outer-string graph if and only if its subdivided apex graph G^+ is a string graph. Furthermore, any outer-string representation R of G can be turned into a string representation of G^+ without changing any curve of R.*

We have two corollaries from this that should be interesting in their own right. First, it is known that every string graph G with m edges has a string representation with $2^{O(m)}$ crossings per string. (This holds since string representations of G correspond to so-called weak realizations of another graph H with $O(m)$ edges [22], and weak realizations can be assumed to have at most 2^m crossings per string [23, 21].) Therefore, if G is outer-string, then the subdivided apex-graph of G has a string representation with $2^{O(m)}$ crossings per string. Deleting the added vertices, we get the following corollary.

▶ **Corollary 2.** *If G is an outer-string graph with m edges, then it has an outer-string representation with $2^{O(m)}$ crossings per string.*

Secondly, while it was long known that string graph recognition is NP-hard [15], proving that it is in NP was a long-standing open question until proved by Schaefer et al. [22]. For any graph G we can construct the subdivided apex graph G^+ in polynomial time. Combining this with Lemma 1 implies the following non-trivial result.

▶ **Corollary 3.** *The problem of recognizing outer-string graphs is in NP.*

Naturally one wonders whether recognizing outer-string graphs is also NP-hard. This problem is open.

2 Exponential-sized Outer-string Representations

Now we construct a graph that requires exponentially many intersections in any string representation. This re-proves a result of Kratochvíl and Matoušek [16], but our graph is different (although inspired by their construction), and can be used to prove the same for outer-string representations later.

For any integer $k \geq 1$ construct graph G_k as follows. For $0 \leq i \leq k$ we add two vertices x_i, y_i, and an edge (x_i, y_j) for every $j \geq i$. We surround this graph with a gadget that forces these vertices to appear in a certain order. This was done in [16] with a grid-like structure, but we use a cycle C instead, in the same way that Cardinal et al. [3] used a cycle to enforce order for their representations.[4] Specifically, let $C = c_0, c_1, \ldots, c_{K-1}$ be a cycle with $K := 8k + 8$ vertices. We connect every 4th vertex of C to one of the vertices $\{x_i, y_i\}_{i=0,\ldots,n}$, in order (along the cycle) $x_0, x_1, y_0, x_2, y_1, x_3, y_2, \ldots, x_i, y_{i-1}, x_{i+1}, y_i, \ldots, x_k, y_{k-1}, y_k$. Let $\ell(x_i)$ [resp. $\ell(y_i)$] be the index of the vertex of C that is adjacent to x_i [resp. y_i], thus $\ell(x_0) = 0$, $\ell(x_1) = 4$, etc. See also Figure 2. This finishes the construction of G_k. As before, let G_k^+ be the subdivided apex graph of G_k with apex vertex a. We use s_j for the subdivision-vertex incident to vertex $c_j \in C$.

Figure 3 illustrates an outer-string representation of G_k, which can be converted into a string representation of its subdivided apex graph G_k^+ (see Lemma 1). Note that $\mathbf{y_k}$ and $\mathbf{x_0}$ intersect 2^{k-1} times. We now argue that this is required.

▶ **Lemma 4.** *In any string representation of G_k^+, curve $\mathbf{y_k}$ intersects curve $\mathbf{x_0}$ at least 2^{k-1} times.*

Proof. Fix a string representation R_k^+ of G_k^+. Delete from it all strings of all subdivision vertices s_{2i+1} for $0 \leq i < K/2$ (these won't be needed). Also, we know that s_{2i} has only two neighbours (a and c_{2i}), and we can hence shorten its string such that $\mathbf{s_{2i}}$ has exactly two intersections, one with $\mathbf{a}$ and one with $\mathbf{c_{2i}}$ [15]. Likewise $\mathbf{c_{2i+1}}$ intersects only two other strings ($\mathbf{c_{2i}}$ and $\mathbf{c_{2i+2}}$) since we deleted $\mathbf{s_{2i+1}}$, and we may hence shorten it such that $|\mathbf{c_{2i}} \cap \mathbf{c_{2i+1}}| = 1 = |\mathbf{c_{2i+1}} \cap \mathbf{c_{2i+2}}|$.

So for any $0 \leq j < K$, we have a unique point in $\mathbf{c_j} \cap \mathbf{c_{j+1}}$ (addition for all vertices in C is mod K). We use $\mathbf{c_j}[c_{j-1}, c_{j+1}]$ to denote the (unique) stretch of $\mathbf{c_j}$ between $\mathbf{c_{j-1}} \cap \mathbf{c_j}$ and $\mathbf{c_j} \cap \mathbf{c_{j+1}}$. Crucial for our argument will be a curve defined by following the strings of cycle C: define $\mathbf{C}$ to be $\bigcup_{j=0}^{K-1} \mathbf{c_j}[c_{j-1}, c_{j+1}]$ and observe that it is a closed simple curve in the plane, hence splits the plane into the inside and outside. We now make a sequence of observations:

- Since the apex-vertex is not adjacent to any vertex of C, curve $\mathbf{a}$ is disjoint from $\mathbf{C}$ and hence resides inside or outside. By symmetry, we may assume that $\mathbf{a}$ is outside $\mathbf{C}$.

[4] The correctness for their gadget was only argued for outer-1-string representations, and so we cannot use it as a black box, but the idea is the same.

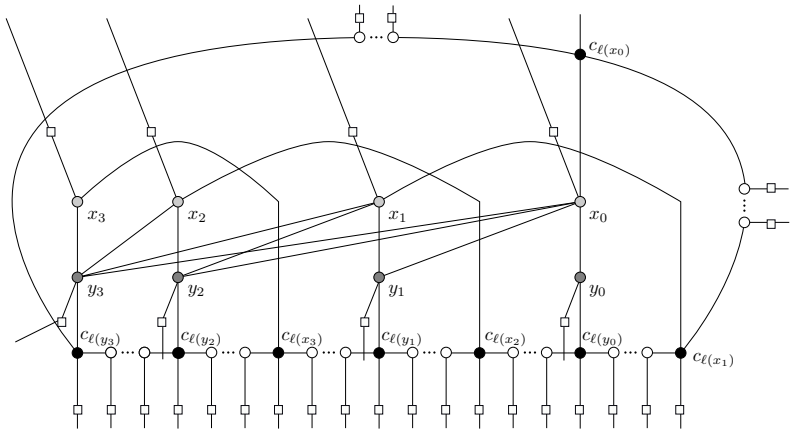


Figure 2 The graph G_3^+. The apex vertex a is not shown. Subdivision vertices are squares.

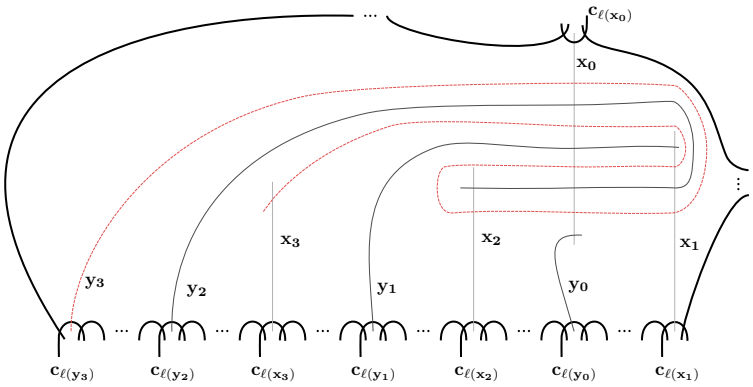


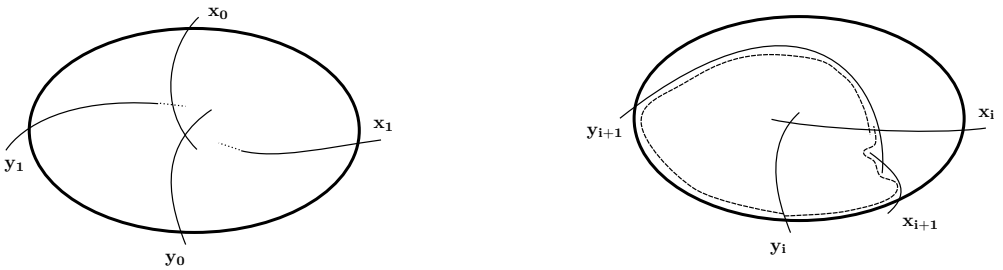
Figure 3 An outer-string representation of G_3. String $\mathbf{y_3}$ is red (dashed) for ease of legibility.

- For any $0 \le i \le k$, curve $\mathbf{x_i}$ has a point outside $\mathbf{C}$. Namely, there exists a subdivision-vertex s_{x_i} with unique neighbors a and x_i. Since neither a nor s_{x_i} have a neighbor on $\mathbf{C}$, and $\mathbf{a}$ is outside $\mathbf{C}$, therefore so is $\mathbf{s_{x_i}}$, and so any point in $\mathbf{s_{x_i}} \cap \mathbf{x_i}$ is outside $\mathbf{C}$.
- For any $0 \le i \le k$, curve $\mathbf{x_i}$ has a point inside $\mathbf{C}$. Specifically, for any $j \ge i$ any point in $\mathbf{x_i} \cap \mathbf{y_j}$ (which exists since there is an edge (x_i, y_j)) must be inside $\mathbf{C}$ for any $j > i$. For otherwise we could use a point in $\mathbf{x_i} \cap \mathbf{y_j}$ outside $\mathbf{C}$ to find a drawing of K_4 with all vertices on one face, an impossibility. (Details are in [6].)
- Thus for any $0 \le i \le k$, curve $\mathbf{x_i}$ has points both inside and outside $\mathbf{C}$. So $\mathbf{x_i}$ must intersect $\mathbf{C}$, which is possible only at $\mathbf{c}_{\ell(\mathbf{x_i})}$.
- Similarly $\mathbf{y_j}$ intersects at a point on $\mathbf{C}$ for all $0 \le j \le k$, and this intersection must happen on $\mathbf{c}_{\ell(\mathbf{y_j})}$.

We are now almost ready to argue the number of intersections of $\mathbf{y_k}$ with $\mathbf{x_0}$, which will happen by induction on k. However, to argue the induction step it helps to permit that some curves do not intersect. We hence use the following type of representation:

▶ **Definition 5.** A *weak outer G_i-representation* is a collection R_i' of curves $\mathbf{C}, \mathbf{x_0}, \mathbf{y_0}, \ldots, \mathbf{x_i}, \mathbf{y_i}$ that satisfies the following:

1. $\mathbf{C}$ is a simple closed curve such that all other curves of R_i' are on or inside $\mathbf{C}$.



(a) The base case.

(b) Two possible routes for $\mathbf{y_i}$.

Figure 4 In the base case, $\mathbf{y_1}$ must cross $\mathbf{x_0}$. In the induction step, a route for $\mathbf{y_{i+1}}$ gives two possible routes for $\mathbf{y_i}$ to $\mathbf{x_i}$.

2. The curves $\mathbf{x_j}$ and $\mathbf{y_j}$ (for $0 \leq j \leq i$) intersect each other.
3. Each of the curves $\mathbf{x_j}$ and $\mathbf{y_j}$ (for $0 \leq j \leq i$) intersects $\mathbf{C}$ exactly once.
4. These intersections with $\mathbf{C}$ occur in order $\mathbf{x_0}$, $\mathbf{x_1}$, $\mathbf{y_0}$, $\mathbf{x_2}$, $\mathbf{y_1}$, $\mathbf{x_3}$, $\mathbf{y_2}, \ldots, \mathbf{x_i}$, $\mathbf{y_{i-1}}$, $\mathbf{y_i}$.
5. The curves $\mathbf{x_r}$ and $\mathbf{y_j}$ (for $0 \leq r < j \leq i$) may or may not intersect each other.
6. No other curves are allowed to intersect each other.

It is easy to see ([6] has details) that for any $0 \leq i \leq k$ we can find a weak outer G_i-representation for which all curves reside within the corresponding curves of R_k^+. The theorem hence holds once we have shown the following:

▶ **Claim 6.** *In any weak outer G_i-representation, curve $\mathbf{y_i}$ intersects $\mathbf{x_0}$ at least 2^{i-1} times.*

We proceed by induction on i. Consider the base case $i = 1$ (see Figure 4(a); for legibility we extend curves slightly beyond $\mathbf{C}$). The order in which curves intersect $\mathbf{C}$ is $\mathbf{x_0}, \mathbf{x_1}, \mathbf{y_0}, \mathbf{y_1}$, and the combined curve $\mathbf{x_0} \cup \mathbf{y_0}$ splits $\mathbf{C}$ into two parts. Curves $\mathbf{x_1}$ and $\mathbf{y_1}$ intersect $\mathbf{C}$ in different parts. To create an intersection point $\mathbf{x_1} \cap \mathbf{y_1}$, one of them must cross paths $\mathbf{y_0} \cup \mathbf{x_0}$. Such a crossing must be between $\mathbf{y_1}$ and $\mathbf{x_0}$ (no other crossings are allowed). So, $\mathbf{y_1}$ intersects $\mathbf{x_0}$ at least once.

Assume now that the claim holds for some i, and study a weak outer G_{i+1}-representation. Curve $\mathbf{y_{i+1}}$ is separated from curve $\mathbf{x_{i+1}}$ by $\mathbf{x_i} \cup \mathbf{y_i}$. Thus, curve $\mathbf{y_{i+1}}$ has to intersect $\mathbf{x_i}$ on its way to $\mathbf{x_{i+1}}$. On the way to $\mathbf{x_i}$, it has to create at least 2^{i-1} intersections with $\mathbf{x_0}$, otherwise we could re-route $\mathbf{y_i}$ and use fewer crossings between $\mathbf{y_i}$ and $\mathbf{x_0}$. More precisely (refer to Figure 4(b)), $\mathbf{y_i}$ could be re-routed to stay in the proximity of the cycle $\mathbf{C}$ until it reaches $\mathbf{y_{i+1}} \cap \mathbf{C}$ and then follow $\mathbf{y_{i+1}}$ until reaching $\mathbf{x_i}$. Along this new route (following $\mathbf{y_{i+1}}$) curve $\mathbf{y_i}$ might intersect neighbors of y_{i+1}, but all those neighbors are allowed to be neighbors of y_i as well, so this is (after deleting $\mathbf{y_{i+1}}$ and $\mathbf{x_{i+1}}$) a weak outer G_i-representation with less than 2^{i-1} points in $\mathbf{y_i} \cap \mathbf{x_0}$. This contradicts the induction hypothesis. So, $\mathbf{y_{i+1}}$ intersects $\mathbf{x_0}$ at least 2^{i-1} times on the way from $\mathbf{C} \cap \mathbf{y_{i+1}}$ to $\mathbf{y_{i+1}} \cap \mathbf{x_i}$.

On the way from $\mathbf{x_i}$ to $\mathbf{x_{i+1}}$, curve $\mathbf{y_{i+1}}$ needs to create another 2^{i-1} crossings with $\mathbf{x_0}$, otherwise we could re-route $\mathbf{y_i}$ and use fewer crossings as follows: $\mathbf{y_i}$ stays in the proximity of the cycle curves until it reaches $\mathbf{x_{i+1}} \cap \mathbf{C}$ and then follows $\mathbf{x_{i+1}}$ and $\mathbf{y_{i+1}}$. Thus $\mathbf{y_{i+1}}$ crosses $\mathbf{x_0}$ at least 2^i times as desired. ◀

In consequence, we have:

▶ **Theorem 7.** *For any $k \geq 1$, there exists a graph G_k with $O(k)$ vertices that has an outer-string representation, but any outer-string representation of G_k requires two strings to intersect at least 2^{k-1} times.*

Proof. We use graph G_k defined earlier; it has $10k + 10$ vertices total. By Lemma 4, any string representation of G_k^+ requires at least 2^{k-1} intersections between y_k and x_0. Since any such representation can be obtained from an outer-string representation of G_k without changing any string of G_k (see Lemma 1), any outer-string representation of G_k requires at least 2^{k-1} intersections between $\mathbf{y_k}$ and $\mathbf{x_0}$. ◀

Since line segments intersect at most once, any polygonal outer-string representation of G_k hence must have a string with at least $\sqrt{2^{k-1}} = 2^{(k-1)/2} \in 2^{\Omega(n)}$ segments.

3 Monotone string representations

In the previous section, we showed that outer-string graphs sometimes require an exponential number of segments in any outer-string representation. Naturally, one wonders whether there are any natural subclasses of string graphs that have polynomial-size representations.

In this section, we prove that there are string representations of polynomial size if the graph is a *monotone string graph*. By this we mean that it has a string representation where every curve is y-monotone, i.e., intersects any horizontal line at most once. Monotone string graphs have been studied before (e.g. in the context of coloring [25]), but to our knowledge the following is new:

▶ **Theorem 8.** *Let G be an n-vertex m-edge graph with a monotone string-representation R. Then G has a monotone string-representation R' with at most $2n(n+m)$ segments.*

Proof. We may assume that no two y-coordinates of crossings or endpoints in R coincide, and no string has its endpoint on another string. Define a *layer-set* $\mathcal{Y}$ of y-coordinates as follows: (1) For every vertex v, add to $\mathcal{Y}$ the y-coordinates of the bottom and top endpoints of $\mathbf{v}$. (2) For every edge $e = (v, w)$, pick one point p in $\mathbf{v} \cap \mathbf{w}$ and add to $\mathcal{Y}$ the y-coordinates $y_e^- := y(p) - \varepsilon$ and $y_e^+ := y(p) + \varepsilon$, where ε is small enough such that no other intersections or endpoints of curves happen within this range. See also Figure 5. Now create R' by defining, for each vertex v, the curve $\mathbf{v}'$ as a poly-line that connects, from bottom to top, the points where $\mathbf{v}$ intersects a horizontal line with y-coordinate in $\mathcal{Y}$. In the rest of the proof we verify that this represents the same graph and satisfies all conditions.

For any $y \in \mathcal{Y}$, define ℓ_Y to be the horizontal line with y-coordinate Y; we call ℓ_Y a *layer*. To define the new curve $\mathbf{v}'$ for a vertex v, let $y_1 < \cdots < y_d$ be all those values $y_i \in \mathcal{Y}$ for which ℓ_{y_i} intersects $\mathbf{v}$. Now let $\mathbf{v}'$ be the poly-line $\mathbf{v} \cap \ell_{y_1}, \mathbf{v} \cap \ell_{y_2}, \ldots, \mathbf{v} \cap \ell_{y_d}$. (These intersection points are unique since $\mathbf{v}$ is monotone.) Curve $\mathbf{v}'$ is monotone and has at most $2m + 2n - 1$ segments.

It remains to argue that R' represents the same graph as R did. If $e = (v, w)$ is an edge, then $\mathbf{v}'$ crosses $\mathbf{w}'$ between the two layers that were added just above and below a point in $\mathbf{v} \cap \mathbf{w}$.

For the other direction, let us assume that curves $\mathbf{v}'$ and $\mathbf{w}'$ cross in R', say at point c. The crossing c cannot lie on a layer ℓ, because both $\mathbf{v}'$ and $\mathbf{w}'$ cross ℓ at the same points as $\mathbf{v}$ and $\mathbf{w}$ did, and $\mathcal{Y}$ was chosen so that no layer contains crossings of R.

So c lies between two consecutive layers, say ℓ and ℓ'. After possible renaming, assume that $\mathbf{v}' \cap \ell$ lies to the left of $\mathbf{w}' \cap \ell$. Since the curves use line segments between layers and there is a crossing, we must have the reverse order on ℓ', i.e., $\mathbf{v}' \cap \ell'$ lies to the right of $\mathbf{w}' \cap \ell'$.

But recall that we chose $\mathbf{v}'$ such that $\mathbf{v}' \cap \ell = \mathbf{v} \cap \ell$, and similarly for w and ℓ'. Therefore, in R we also had $\mathbf{v}$ to the left of $\mathbf{w}$ on ℓ and to the right of $\mathbf{w}$ on ℓ'. Curves $\mathbf{v}$ and $\mathbf{w}$ are y-monotone in the stretch between ℓ and ℓ'. It follows that the two curves $\mathbf{v}$ and $\mathbf{w}$ cross somewhere within this stretch. Therefore (v, w) is an edge of the graph as required. ◀

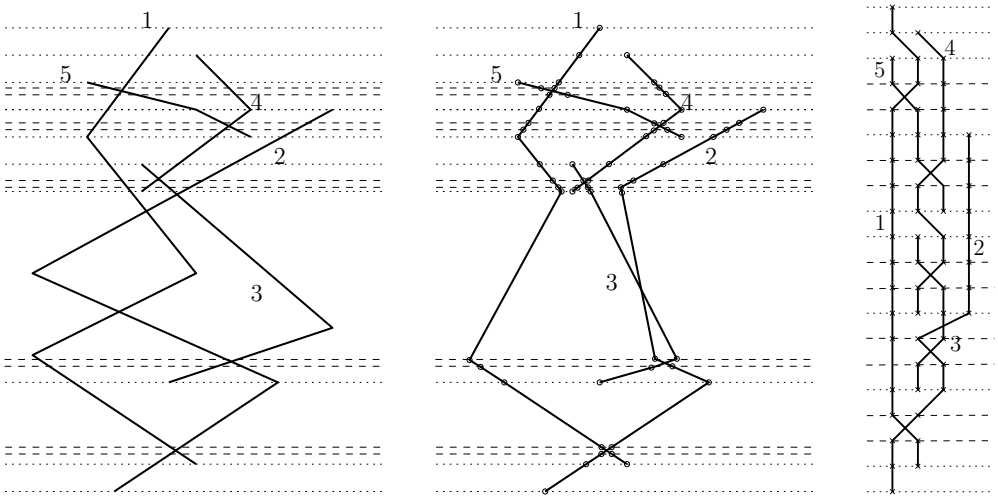


Figure 5 A monotone string-representation of C_5, an application of our algorithm, and re-assigning coordinates to obtain an $n \times (2m + 2n)$-grid.

We note that R' can be assumed to reside on an $n \times (2m + n)$-grid. Namely, each curve consists of line segments that connect consecutive layers. We can re-assign y-coordinates in $\{1, \ldots, 2m + 2n\}$ to the layers, and re-assign x-coordinates in $\{1, \ldots, n\}$ to the points where curves intersect layers, and the same line segments will cross between consecutive layers, hence we obtain a string representation of the same graph. See Figure 5.

One drawback of our proof is that it needs an explicit representation R to create the polynomial-sized representation R'. It remains open how to find such a representation R', given just the graph.

4 Independent set in monotone outer-string graphs

Keil et al. presented an algorithm for (weighted) independent set on outer-string graphs that runs in time $O(N^3)$ (as before, N is the size of an outer-string representation) [14]. However, due to Theorem 7, N may need to be in $2^{\Omega(n)}$. In this section we study the independent set problem on monotone string graphs, which have a polynomial-size representation by Theorem 8. Since planar graphs are segment graphs [4] (hence monotone string graphs), and since maximum independent set is NP-hard for planar graphs [11], we have:

▶ **Proposition 9.** *Maximum independent set is NP-hard even for monotone string graphs.*

We therefore turn our attention to monotone outer-string graphs. Here, we know from Keil et al.'s result that the maximum independent set problem is solvable in polynomial time in the size of representation, and from Theorem 8 that there exists a representation with size $N \in O(nm)$ and at most $O(m + n)$ line segments per string. Presuming such a representation is given, we can hence solve the independent set problem in $O(n^3m^3)$ time. We now show that for two special cases of monotone outer-string graphs, a better run-time can be achieved.

▶ **Definition 10.** Let G be a monotone outer-string graph. We say that G is *strip-grounded* if there exists a monotone string representation of G with a bounding rectangle ρ such that all strings have one end at the top or bottom side of ρ. We say that G is *line-grounded* if all strings have one end on the bottom side of ρ.

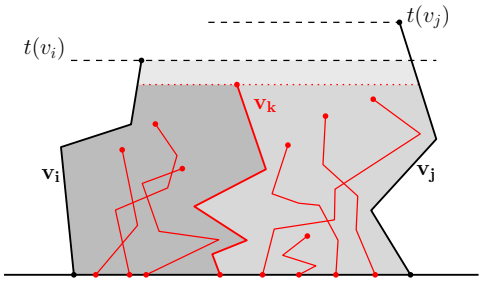


Figure 6 Line-grounded strings, and an illustration of the formula for $W(i,j)$.

Figures 6 and 7 illustrate line-grounded and strip-grounded graphs, respectively. We may, after shortening some strings, assume that no string in such a representation touches both the bottom and the top of ρ. For a string $\mathbf{v}$, we use $b(v)$ and $t(v)$ to denote the y-coordinates of the bottom and top endpoints of $\mathbf{v}$, respectively. For ease of description, we add two negative-weight dummy vertices with strings along the left and right sides of ρ (no optimal solution will include these two vertices/strings). Enumerate the *bottom-grounded vertices* (i.e., vertices whose strings attach at the bottom side of ρ) as $v_1, \ldots, v_b$, from left to right by bottom endpoint. Enumerate the *top-ground vertices* as $u_1, \ldots, u_t$, from left to right by top endpoint. Here, $v_1 = u_1$ and $v_b = u_t$ are the dummy vertices.

4.1 Line-grounded monotone string graphs

We first show how to find the maximum independent set in a line-grounded monotone string graph G; this will be a useful subroutine later. We only have vertices $v_1, \ldots, v_b$ (with $b = n + 2$ due to the dummy vertices). We proceed by dynamic programming, and define sub-problems as follows (a similar technique has been used in [1] for computing approximate maximum independent set of segments that cross a straight line). For any pair (i,j), with $1 \le i < j \le b$ and $(v_i, v_j) \notin E$, define $S(i,j)$ to be the set of vertices $v_k \in \{v_{i+1}, \ldots, v_{j-1}\}$ that satisfy $t(v_k) \le \min\{t(v_i), t(v_j)\}$, $(v_k, v_i) \notin E$ and $(v_k, v_j) \notin E$. Put differently, $S(i,j)$ contains every vertex v_k for which $\mathbf{v_k}$ is strictly within the region bounded by $\mathbf{v_i}$, the bottom side of ρ, $\mathbf{v_j}$, and the horizontal line with y-coordinate $\min\{t(v_i), t(v_j)\}$ (see Figure 6). Due to the dummy vertices, we have $S(1,b) = V$.

Let $w(v)$ be the weight of vertex v, and set $W(i,j)$ to be the weight of a maximum independent set in $S(i,j)$.

▶ **Claim 11.** $W(i,j) = \begin{cases} 0 & \text{if } S(i,j) \text{ is empty} \\ \max_{v_k \in S(i,j)} W(i,k) + W(k,j) + w(v_k) & \text{otherwise.} \end{cases}$

Proof. See Figure 6 for an illustration of this proof. Consider an optimal solution I^* for $S(i,j)$. Let v_k be the vertex that maximizes $t(v_k)$ among the vertices in I^* (if there is no such v_k then $S(i,j) = \emptyset$ and the equality holds). Let v be some other vertex in I^*. Since I^* is an independent set, $\mathbf{v}$ does not intersect $\mathbf{v_k}$. It also intersects neither $\mathbf{v_i}$ nor $\mathbf{v_j}$ by definition of $S(i,j)$. Finally $t(v) \le t(v_k)$ by choice of v_k. It follows that $v \in S(i,k)$ or $v \in S(k,j)$. So $I^* - \{v_k\}$ induces two independent sets for $S(i,k)$ and $S(k,j)$. So "$\le$" holds for this choice of v_k, and even more so for the maximum among all v_k in $S(i,j)$.

For the other direction, let k be the index where the maximum is achieved and fix maximum independent sets I_i and I_j of $S(i,k)$ and $S(k,j)$. Observe that no string of I_i can intersect one in I_j since they reside within disjoint regions, and neither of them can intersect $\mathbf{v_k}$ by definition of $S(i,k)$ and $S(k,j)$. So $I_i \cup I_j \cup \{v_k\}$ is an independent set of $S(i,j)$ and "$\ge$" holds. ◀

By computing $S(1, b)$ recursively with standard dynamic programming techniques, we can hence find the maximum independent set of G. We briefly discuss the run time. To find set $S(i, j)$, we mark all neighbours of v_i, all neighbours of v_j, and all vertices v with $t(v) > \min\{t(v_i), t(v_j)\}$. Then we take all unmarked vertices in $\{v_{i+1}, \ldots, v_{j-1}\}$; clearly this takes $O(n)$ time per set $S(i, j)$. Evaluating the recursive formula takes $O(n)$ time as well, and since we have $O(n^2)$ subproblems, the overall run-time is $O(n^3)$. (Note that for this algorithm, we do not even need an explicit line-grounded monotone string representation: it suffices to have graph G, and the coordinates of the top and bottom endpoints, together with the promise that they correspond to such a representation.)

▶ **Theorem 12.** *Given a vertex-weighted graph G with a line-grounded monotone string representation, we can compute the maximum-weight independent set of G in $O(n^3)$ time.*

4.2 Strip-grounded monotone string graphs

Now we turn to strip-grounded monotone string graphs. First note that by applying the algorithm for line-grounded monotone string graphs twice (once for the bottom-grounded vertices and once for the top-grounded vertices), we immediately obtain a 2-approximation algorithm, which runs in $O(n^3)$ time. At the price of an increased run-time, we show how to solve this problem optimally. For this, we need a more complicated set of subproblems:

Let $\mathbf{v}$ be a bottom-grounded string and $\mathbf{u}$ be a top-grounded string such that $(v, u) \notin E$ and $t(v) \geq b(u)$. We say that a vertex x lies *between v and u* if there exists a horizontal line ℓ that intersects all of $\mathbf{v}, \mathbf{u}, \mathbf{x}$, and for which the point $\ell \cap \mathbf{x}$ lies between the points $\ell \cap \mathbf{v}$ and $\ell \cap \mathbf{u}$. Define the following sets (see also Figure 7):

- Let $1 \leq i \leq j \leq b$ and $1 \leq \alpha \leq \beta \leq t$ be indices such that $\{v_i, u_\alpha, v_j, u_\beta\}$ is an independent set, and further $t(v_i) \geq b(u_\alpha)$ and $t(v_j) \geq b(u_\beta)$. Define $S(i, \alpha; j, \beta)$ to be all those vertices in $\{v_{i+1}, \ldots, v_{j-1}\} \cup \{u_{\alpha+1}, \ldots, u_{\beta-1}\}$ that are adjacent to none of $v_i, v_j, u_\alpha, u_\beta$, and do not lie between v_i and u_α or between v_j and u_β.
- Let $1 \leq i \leq k \leq b$ and $1 \leq \alpha \leq t$ be indices such that $\{v_i, u_\alpha, v_k\}$ is an independent set and $t(v_i) \geq b(u_\alpha)$. Define $S^{SW}(i, \alpha; k)$ to be all those vertices v in $\{v_{i+1}, \ldots, v_{k-1}\}$ that are adjacent to none of v_i, v_k, u_α, do not lie between v_i and u_α, and for which $t(v) \leq t(v_k)$.
- We symmetrically define $S^{SE}(k; j, \beta)$, $S^{NW}(i, \alpha; \gamma)$ and $S^{SW}(\gamma; j, \beta)$. See also Figure 7.
- Finally we also need the set $S(i, j)$ defined earlier (we denote it $S_v(i, j)$ since it uses the bottom-grounded vertices), and symmetrically set $S_u(\alpha, \beta)$ for top-grounded vertices.

Let $W(i, j; \alpha, \beta)$ be the weight of a maximum independent set in subgraph induced by vertex set $S(i, \alpha; j, \beta)$, and similarly for all other sets. We already had the formula for $W_v(i, j)$ (Claim 11), and a symmetric one holds for $W_u(\alpha, \beta)$. With much the same proof one can show (see also Figure 7(c)):

▶ **Claim 13.** *$W^{SW}(i, k; \alpha) = 0$ if $S^{SW}(i, k; \alpha)$ is empty. Otherwise,*

$$W^{SW}(i, k; \alpha) = \max_{v_r \in S^{SW}(i,k;\alpha)} W^{SW}(i, r; \alpha) + W_v(r, k) + w(v_k).$$

The formula for W^{SW}, W^{NE} and W^{NW} are symmetric. As for $W(i, j; \alpha, \beta)$, based on whether the maximum independent set contains bottom-grounded or top-grounded vertices, and how they interact, one can show the following formula:

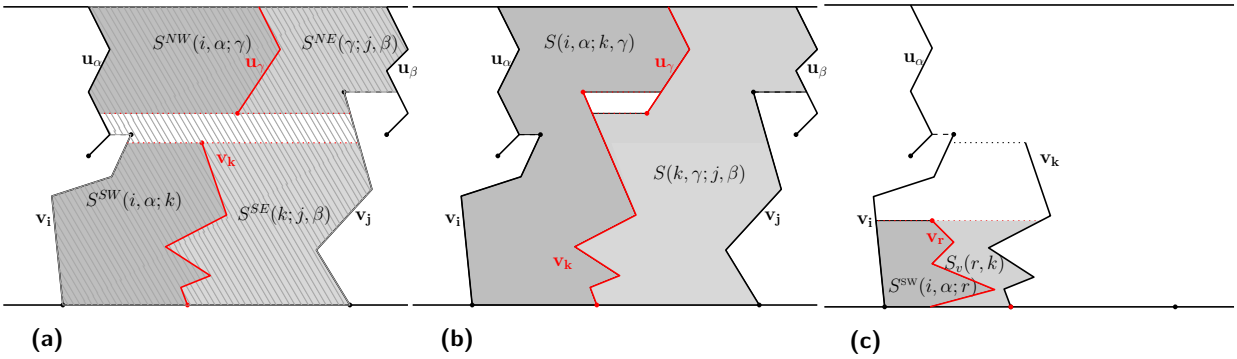


Figure 7 A strip-grounded graph. Strings in $S(i,\alpha;j,\beta)$ must be in the striped region. We illustrate recursive formulas (a) for $W(i,\alpha,j,k)$ for $t(v_k) < b(u_\gamma)$, (b) for $W(i,\alpha,j,k)$ for $t(v_k) \geq b(u_\gamma)$, and (c) for $W^{SW}(i,\alpha;k)$.

▶ **Claim 14.** *$W(i,j;\alpha,\beta) = 0$ if $S(i,\alpha;j,\beta)$ is empty. Otherwise, it is the maximum of*

$$\begin{aligned}
&\max_{v_k \in S(i,\alpha;j,\beta)} && W^{SW}(i,\alpha;k) + W^{SE}(k;j,\beta) + w(v_k),\\
&\max_{u_\gamma \in S(i,\alpha;j,\beta)} && W^{NW}(i,\alpha;\gamma) + W^{NE}(\gamma;j,\beta) + w(u_\gamma),\\
&\max_{v_k,u_\gamma \in S(i,\alpha;j,\beta), t(v_k)<b(u_\gamma)} && W^{SW}(i,\alpha;k) + W^{SE}(k;j,\beta) + w(v_k)\\
& && +W^{NW}(i,\alpha;\gamma) + W^{NE}(\gamma;j,\beta) + w(u_\gamma), \text{ and}\\
&\max_{v_k,u_\gamma \in S(i,\alpha;j,\beta), t(v_k)>b(u_\gamma), (v_k,u_\gamma)\notin E} && W(i,k;\alpha,\gamma) + W(k,j;\gamma,\beta) + w(v_k) + w(u_\gamma)
\end{aligned}$$

Proof. To show '$\geq$', observe that each term of the maximum on the right-hand side corresponds to two or four independent sets in two or four regions defined by the parameters. As one easily verifies, these regions are disjoint for all cases, and none of them contains $\mathbf{v_k}$ and/or $\mathbf{u}_\gamma$. We can hence combine these independent sets and add v_k and/or u_γ, and obtain an independent set for $S(i,\alpha;j,\beta)$. The optimum independent set cannot be smaller.

To prove '$\leq$', consider an optimal solution I^* for $S(i,\alpha;j,\beta)$. We may assume that I^* is non-empty; else $S(i,\alpha;j,\beta)$ is empty and the equation holds. We distinguish cases:

Case 1: I^* contains no top-grounded vertex. Since I^* is non-empty, it therefore contains some v_k with $i < k < j$. Let v_k be the vertex that maximizes $t(v_k)$. With this choice, any other vertex in I^* is bottom-grounded and belongs to $S^{SW}(i,\alpha;k)$ or $S^{SE}(k;j,\beta)$, depending on whether its index is before or after k. Therefore $I^* - \{v_k\}$ splits into two independent sets for $S^{SW}(i,\alpha;k)$ and $S^{SE}(k;j,\beta)$, and $w(I^*) + w(v_k) \leq W^{SW}(i,\alpha;k) + W^{SE}(k;j,\beta)$.

Case 2: I^* contains no bottom-grounded vertex. Symmetrically then one shows that $w(I^*) \leq W^{NW}(i,\alpha;\gamma) + W^{NE}(\gamma;j,\beta) + w(u_\gamma)$ where u_γ is the top-grounded vertex in I^* that minimizes $b(u_\gamma)$.

Case 3: I^* contains a bottom-grounded vertex v_k and a top-grounded vertex u_γ, but for any two such vertices we have $t(v_k) < b(u_\gamma)$. Choose v_k so that it maximizes $t(v_k)$ and u_γ so that it minimizes $b(u_\gamma)$. Then any other bottom-grounded vertex v in I^* satisfies $t(v) \leq t(v_k)$ and so belongs to $S^{SW}(i,\alpha;k)$ or $S^{SE}(k;j,\beta)$. Any other top-grounded vertex u in I^* satisfies $b(u) \geq b(u_\gamma)$ and so belongs to $S^{NE}(i,\alpha;\gamma)$ or $S^{NE}(\gamma;j,\beta)$. Thus $I^* - \{v_k, u_\gamma\}$ splits into four independent sets for these four vertex sets, hence $w(I^*) \leq W^{SW}(i,\alpha;k) + W^{SE}(k;j,\beta) + w(v_k) + W^{NW}(i,\alpha;\gamma) + W^{NE}(\gamma;j,\beta) + w(u_\gamma)$.

Case 4: I^* contains a bottom-grounded vertex v_k and a top-grounded vertex u_γ with $t(v_k) \geq b(u_\gamma)$. Thus, any line ℓ with y-coordinate in $[b(u_\gamma), t(v_k)]$ intersects both $\mathbf{v_k}$ and $\mathbf{u_\gamma}$. We may assume that any such line ℓ intersects no other string of I^* in the range between $\ell \cap \mathbf{v_k}$ and $\ell \cap \mathbf{u_\gamma}$, else we can replace either v_k or u_γ with the intersected string. Thus no other vertex x in I^* is between v_k and u_γ. Therefore any vertex $x \neq v_k, u_\gamma$ in I^* belongs to either $S(i, \alpha; k, \gamma)$ or to $S(k, \gamma; j, \beta)$. So $I^* - \{v_k, u_\gamma\}$ splits into two independent sets for these two subsets. This proves that $w(I^*) \leq W(i, \alpha; k, \gamma) + W(k, \gamma; j, \beta) + w(v_k) + w(u_\gamma)$. ◀

Since $S(1, a; 1, b) = V$ with our choice of dummy vertices, therefore we can compute the maximum independent set in G with dynamic programming. To analyze its run-time, observe that we have defined $O(n^4)$ sets. To compute each set, we need to test quickly whether x is between v_i and u_α for some independent set $\{x, v_i, u_\alpha\}$. We first test whether there exists some Y with $b(u_\alpha) \leq Y \leq t(v_i)$ and $b(x) \leq Y \leq t(x)$; otherwise x is surely not between them. If there is such a Y, then next find the points p_u, p_v, p_x where the horizontal line with y-coordinate Y intersects the three strings, and test their order. Recall that we assumed the strings to have $O(m + n)$ segments, so finding these points (and hence testing whether x is between v_i and u_α) can be done with binary search in $O(\log n)$ time.

With this, each set can be found in $O(n^2)$ time. For example, to find $S(i, \alpha; j, \beta)$, scan the vertices $v_{i+1}, \ldots, v_{j-1}$ and $u_{\alpha+1}, \ldots, u_{\beta-1}$. For each, test in $O(n)$ time whether it is non-adjacent to $v_i, u_\alpha, v_j, u_\beta$, and test in $O(\log n)$ time that it is neither between v_i and u_α, nor between v_j and u_β. The computation for the other types of sets is similar.

Given all the sets, the evaluation of the formula can be done in $O(n^2)$ time per set, or $O(n^6)$ total for all $O(n^4)$ sets. Therefore, we get the following theorem.

▶ **Theorem 15.** *The maximum independent set in a vertex-weighted graph with a strip-grounded monotone string representation can be computed in $O(n^6)$ time.*

5 Conclusions

In this paper, we studied graphs that do or do not have string representations of polynomial size. We argued that for some outer-string graphs any outer-string representation must have exponential size. On the other hand, all monotone string graphs have a string representation of polynomial size. Inspired by an algorithm of Keil et al. for maximum independent set for outer-string graphs, we give an algorithm for maximum independent set for monotone strip-grounded outer-string graphs, whose run-time is $O(n^6)$, presuming we are given such a representation of polynomial size. We leave a number of open problems:

- We have introduced some variants of string graphs (e.g., monotone string graphs, monotone strip-grounded string graphs). What is the complexity of recognizing these graphs classes and finding corresponding representations? Note that it is not even known whether recognizing outer-string graphs is NP-hard (we proved that it is in NP).
- What is the complexity of recognizing whether a graph has a string representation (or outer-string representation) with at most k segments?
- Is there an algorithm for independent set on outer-string graphs that is polynomial in n? Our results show that this is not possible if we use an explicit description of a string representation. But perhaps the string representation could be given implicitly in a different way? Or perhaps it could be described (similarly as in [22]) with $O(\log N)$ bits, by listing how it intersects a (suitably) chosen triangulation? Note that $\log N \in O(m)$, so this would be polynomial.

References

1. Pankaj K. Agarwal and Nabil H. Mustafa. Independent set of intersection graphs of convex objects in 2D. *Comput. Geom.*, 34(2):83–95, 2006.
2. Sergio Cabello and Miha Jejcic. Refining the hierarchies of classes of geometric intersection graphs. *Electronic Notes in Discrete Mathematics*, 54:223–228, 2016. doi:10.1016/j.endm.2016.09.039.
3. Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber. Intersection graphs of rays and grounded segments. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, volume 10520 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2017. doi:10.1007/978-3-319-68705-6_12.
4. Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 631–638. ACM, 2009. doi:10.1145/1536414.1536500.
5. Peter Damaschke. The hamiltonian circuit problem for circle graphs is np-complete. *Inf. Process. Lett.*, 32(1):1–2, 1989. doi:10.1016/0020-0190(89)90059-8.
6. Martin Derka. *Restricted String Representations*. PhD thesis, David R. Cheriton School of Computer Science, 2017. URL: https://uwspace.uwaterloo.ca/handle/10012/12253.
7. Gideon Ehrlich, Shimon Even, and Robert Endre Tarjan. Intersection graphs of curves in the plane. *J. Comb. Theory, Ser. B*, 21(1):8–20, 1976. doi:10.1016/0095-8956(76)90022-8.
8. Jacob Fox and János Pach. Separator theorems and Turán-type results for planar intersection graphs. *Adv. Math.*, 219:1070–1080, 2008.
9. Jacob Fox and János Pach. A separator theorem for string graphs and its applications. *Combinatorics, Probability & Computing*, 19(3):371–390, 2010. doi:10.1017/S0963548309990459.
10. Jacob Fox and János Pach. Computing the independence number of intersection graphs. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1161–1165. SIAM, 2011. doi:10.1137/1.9781611973082.87.
11. M. R. Garey and David S. Johnson. The rectilinear steiner tree problem in NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.
12. M. R. Garey, David S. Johnson, G. L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Matrix Analysis Applications*, 1(2):216–227, 1980. doi:10.1137/0601025.
13. Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 717–728. Springer, 2015. doi:10.1007/978-3-662-48350-3_60.
14. J. Mark Keil, Joseph S. B. Mitchell, Dinabandhu Pradhan, and Martin Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Comput. Geom.*, 60:19–25, 2017. Appeared also in the Proceedings of CCCG 2015.
15. Jan Kratochvíl. String graphs. II. recognizing string graphs is np-hard. *J. Comb. Theory, Ser. B*, 52(1):67–78, 1991. doi:10.1016/0095-8956(91)90091-W.
16. Jan Kratochvíl and Jiří Matoušek. String graphs requiring exponential representations. *J. Comb. Theory, Ser. B*, 53(1):1–4, 1991. doi:10.1016/0095-8956(91)90050-T.

17 James R. Lee. Separators in region intersection graphs. In *Innovations in Theoretical Computer Science, ITCS'17*, 2017.

18 Jiří Matoušek. Near-optimal separators in string graphs. *CoRR*, abs/1302.6482, 2013. arXiv:1302.6482.

19 Matthias Middendorf and Frank Pfeiffer. The max clique problem in classes of string-graphs. *Discrete Mathematics*, 108(1-3):365–372, 1992. doi:10.1016/0012-365X(92)90688-C.

20 Matthias Middendorf and Frank Pfeiffer. Weakly transitive orientations, hasse diagrams and string graphs. *Discrete Mathematics*, 111(1-3):393–400, 1993. doi:10.1016/0012-365X(93)90176-T.

21 János Pach and Géza Tóth. Recognizing string graphs is decidable. *Discrete & Computational Geometry*, 28(4):593–606, 2002. doi:10.1007/s00454-002-2891-4.

22 Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. Recognizing string graphs is in NP. *Journal of Computer and System Sciences*, 67(2):365–380, 2003.

23 Marcus Schaefer and Daniel Stefankovic. Decidability of string graphs. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 241–246. ACM, 2001. doi:10.1145/380752.380807.

24 F. W. Sinden. Topology of thin film rc-circuits. *Bell System Technical Journal*, 45:1639–1662, 1966. doi:10.1002/j.1538-7305.1966.tb01713.x.

25 Andrew Suk. Coloring intersection graphs of x-monotone curves in the plane. *Combinatorica*, 34(4):487–505, 2014. doi:10.1007/s00493-014-2942-5.

Flip Distance to some Plane Configurations

Ahmad Biniaz[1]
Cheriton School of Computer Science, University of Waterloo
Waterloo, Canada
ahmad.biniaz@gmail.com

Anil Maheshwari[2]
School of Computer Science, Carleton University
Ottawa, Canada
anil@scs.carleton.ca

Michiel Smid[3]
School of Computer Science, Carleton University
Ottawa, Canada
michiel@scs.carleton.ca

Abstract

We study an old geometric optimization problem in the plane. Given a perfect matching M on a set of n points in the plane, we can transform it to a non-crossing perfect matching by a finite sequence of flip operations. The flip operation removes two crossing edges from M and adds two non-crossing edges. Let $f(M)$ and $F(M)$ denote the minimum and maximum lengths of a flip sequence on M, respectively. It has been proved by Bonnet and Miltzow (2016) that $f(M) = O(n^2)$ and by van Leeuwen and Schoone (1980) that $F(M) = O(n^3)$. We prove that $f(M) = O(n\Delta)$ where Δ is the spread of the point set, which is defined as the ratio between the longest and the shortest pairwise distances. This improves the previous bound for point sets with sublinear spread. For a matching M on n points in convex position we prove that $f(M) = n/2-1$ and $F(M) = \binom{n/2}{2}$; these bounds are tight.

Any bound on $F(\cdot)$ carries over to the bichromatic setting, while this is not necessarily true for $f(\cdot)$. Let M' be a bichromatic matching. The best known upper bound for $f(M')$ is the same as for $F(M')$, which is essentially $O(n^3)$. We prove that $f(M') \leqslant n - 2$ for points in convex position, and $f(M') = O(n^2)$ for semi-collinear points.

The flip operation can also be defined on spanning trees. For a spanning tree T on a convex point set we show that $f(T) = O(n \log n)$.

2012 ACM Subject Classification Theory of computation → Computational geometry, Mathematics of computing → Discrete mathematics

Keywords and phrases flip distance, non-crossing edges, perfect matchings, spanning trees

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.11

1 Introduction

A *geometric graph* is a graph whose vertices are points in the plane, and whose edges are straight-line segments connecting the points. All graphs that we consider in this paper are geometric. A graph is *plane* if no pair of its edges cross each other. Let $n \geqslant 2$ be an even integer, and let P be a set of n points in the plane that is in general position (no three points

[1] Supported by NSERC and Fields Institute.
[2] Supported by NSERC.
[3] Supported by NSERC.

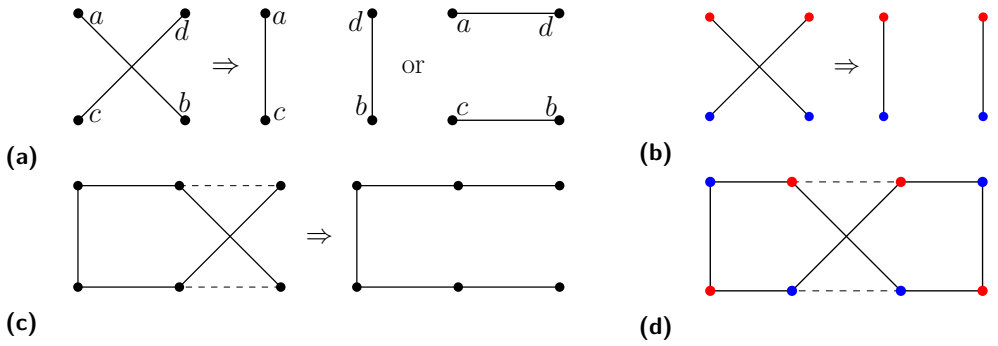


Figure 1 (a) Two ways to flip a crossing in a monochromatic matching. (b) The only way to flip a crossing in a bichromatic matching. (c) One way to flip a crossing in a monochromatic tree. (d) No way to flip a crossing in a bichromatic Hamiltonian cycle.

on a line). For two points a and b in the plane, we denote by ab the segment with endpoints a and b. Let M be a perfect matching on P. If two edges in M cross each other, we can remove this crossing by a flip operation. The *flip operation* (or flip for short) removes two crossing edges and adds two non-crossing edges to obtain a new perfect matching. In other words, if two segments ab and cd cross, then a flip removes ab and cd from the matching, and adds either ac and bd, or ad and bc to the matching; see Figure 1(a). Every flip decreases the total length of the edges of M, and thus, after a finite sequence of flips, M can be transformed to a plane perfect matching. This process of transforming a crossing matching to a plane matching is referred to as *uncrossing* or *untangling* a matching. Motivated by this old folklore result, we investigate the minimum and the maximum lengths of a sequence of flips to reach a plane matching.

To uncross a perfect matching M, we say that the sequence $(M{=}M_0, M_1, \ldots, M_k)$ is a valid flip sequence if M_{i+1} is obtained from M_i by a single flip, and M_k is plane. The number k denotes the length of this flip sequence. We define $f(M)$ to be the minimum length of any valid flip sequence to uncross M, that is, the minimum number of flips required to transform M to a plane perfect matching. We define $F(M)$ to be the maximum length of any valid flip sequence. As for $F(M)$, one can imagine that an adversary imposes which of the two flips to apply on which of the crossings.

In the bichromatic setting, we are given $n/2$ red and $n/2$ blue points and a bichromatic matching, that is a perfect matching in which the two endpoints of every segment have distinct colors. Contrary to the monochromatic setting, there is only one way to flip two crossing bichromatic edges; see Figure 1(b). In the bichromatic setting the adversary can only impose the crossing to flip. Thus, any upper bound on $F(M)$ for monochromatic matchings carries over to bichromatic matchings; this statement is not necessarily true for $f(M)$.

The flip operation can be defined for a spanning tree (resp. a Hamiltonian cycle) analogously, that is, we remove a pair of crossing edges and add two other edges so that the graph remains a spanning tree (resp. a Hamiltonian cycle) after this operation. We define $f(\cdot)$ and $F(\cdot)$ for spanning trees and Hamiltonian cycles, analogously. As shown in Figure 1(c), there is only one way to flip a crossing in a spanning tree (resp. a Hamiltonian cycle). Contrary to the bichromatic matching, it is not always possible to flip a crossing in a bichromatic spanning tree nor in a bichromatic Hamiltonian cycle; see Figure 1(d).

1.1 Related Work

The most relevant works are by van Leeuwen and Schoone [21], and Oda and Watanabe [17] for Hamiltonian cycles, and by Bonnet and Miltzow [7] for matchings. They proved, with elegant arguments, the following results.

▶ **Theorem 1** (van Leeuwen and Schoone, 1981 [21]). *For every Hamiltonian cycle H on n points in the plane we have that $F(H) = O(n^3)$.*

▶ **Theorem 2** (Oda and Watanabe, 2007 [17]). *For every Hamiltonian cycle H on n points in the plane in convex position we have that $f(H) \leqslant 2n - 7$.*

As for a lower bound, they presented a Hamiltonian cycle H on $n \geqslant 7$ points in the plane in convex position for which $f(H) \geqslant n - 2$.

▶ **Theorem 3** (Bonnet and Miltzow, 2016 [7]). *For every perfect matching M on a set of n points in the plane in general position we have that $f(M) = O(n^2)$.*

The $O(n^3)$ upper bound of Theorem 1 carries over to perfect matchings. As for lower bounds, Bonnet and Miltzow [7] presented two matchings M_1 and M_2 such that $f(M_1) = \Omega(n)$ and $F(M_2) = \Omega(n^2)$. The bound $F(M) = O(n^3)$ holds even if M is a bichromatic matching, while the proof of $f(M) = O(n^2)$ does not generalize for the bichromatic setting.

An alternate definition of an *edge flip* in a graph is the operation of removing one edge and inserting a different edge such that the resulting graph remains in the same graph class. The edge flip operation has been studied for many different graph classes, in particular, for two given graphs with an equal number of vertices and edges, the number of edge flips required to transform one into another. See the survey by Bose and Hurtado [8] on edge flips in planar graphs both in the combinatorial and the geometric settings, and see [3, 9, 14, 16] for edge flips in triangulations.

A related problem is the compatible matching problem in which we are given two perfect matchings on the same point set and the goal is to transform one to another by a sequence of compatible matchings (two perfect matchings, on the same point set, are said to be compatible if they are edge disjoint and their union is non-crossing). See [1, 2, 4, 15] for recent work on compatible matchings, and [18] for its extension to compatible trees.

1.2 Our Contribution

In this paper we decrease the gap between lower and upper bounds for $f(\cdot)$ and $F(\cdot)$ for some input configurations. In Section 2 we show that for every perfect matching M, on a set P of n points in the plane, we have $f(M) = O(n\Delta)$ where Δ is the spread of P.

Assume that P is in convex position. In Section 3 we show that for every perfect matching M on P we have that $f(M) \leqslant n/2 - 1$ and $F(M) \leqslant \binom{n/2}{2}$. These bounds are tight as Bonnet and Miltzow [7] showed the existence of two perfect matchings M_1 and M_2 on n points in convex position such that $f(M_1) \geqslant n/2 - 1$ and $F(M_2) \geqslant \binom{n/2}{2}$. We also prove that for every spanning tree T on P we have that $f(T) = O(n \log n)$.

In Section 4 we study bichromatic matchings on special point sets. Assume that the points of P are colored red and blue. We prove that, if P is in convex position, then for every perfect bichromatic matching M on P we have that $f(M) \leqslant n - 2$. Also, we prove that, if P is semi-collinear, i.e., the blue points are on a straight line, then for every perfect bichromatic matching M on P we have that $f(M) = O(n^2)$. Table 1 summarizes the results.

Table 1 Upper bounds on the minimum and maximum number of flips (Δ is the spread).

minimum # of flips	$f(\cdot)$-general position		$f(\cdot)$-convex position	
matchings	$O(n^2)$	[7]	$n/2-1$	Theorem 11
	$O(n\Delta)$	Theorem 4		
bichromatic matchings	$O(n^3)$	[21]	$n-2$	Theorem 15
trees	$O(n^3)$	[21]	$O(n \log n)$	Theorem 14
Hamiltonian cycles	$O(n^3)$	[21]	$2n-7$	[17]
bichromatic matching on semi-collinear points	$f(\cdot) = O(n^2)$			Theorem 17

maximum # of flips	$F(\cdot)$-general position		$F(\cdot)$-convex position	
matchings/trees/cycles	$O(n^3)$	[21]	$\binom{n/2}{2}$	Theorem 11

1.3 Preliminaries

Let a and b be two points in the plane. We denote by ab the straight line-segment between a and b, and by $\overline{ab}$ the line through a and b. Let P be a set of points in the plane in convex position. For two points p and q in P we define the *depth* of the segment pq as the minimum number of points of $P \setminus \{p, q\}$ on either side of $\overline{pq}$. A *boundary edge* is a segment of depth zero, i.e., an edge of the convex hull of P. An edge e in a graph G is said to be *free* if e is not crossed by other edges of G.

2 Minimum Number of Flips

The *spread* Δ of a set of points (also called the *distance ratio* [11]) is the ratio between the largest and the smallest interpoint distances. It is well known that the spread of a set of n points in the plane is $\Omega(\sqrt{n})$ (see e.g., [19]). In this section, we prove an upper bound on the minimum length of a flip sequence in terms of n and Δ. In fact we prove the following theorem.

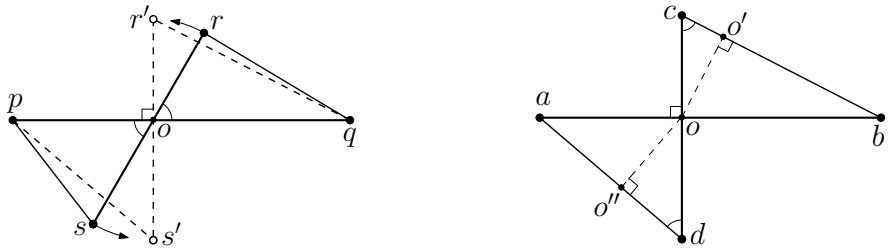
▶ **Theorem 4.** *For every perfect matching M on a set of n points in the plane in general position we have that $f(M) = O(n\Delta)$, where Δ is the spread of the point set.*

For point sets with spread $o(n)$, the upper bound of Theorem 4 is better than the $O(n^2)$ upper bound of Theorem 3. For example, for *dense* point sets, which have spread $O(\sqrt{n})$, Theorem 4 gives an upper bound of $O(n\sqrt{n})$ on the number of flips. According to [13], dense point sets commonly appear in nature, and they have applications in computer graphics. Valtr and others [13, 19, 20] have established several combinatorial bounds for dense point sets that improve corresponding bounds for arbitrary point sets.

Let P be a set of n points in the plane with spread Δ. Let M be a perfect matching on P. We prove that M can be untangled by $O(n\Delta)$ flips, i.e., $f(M) = O(n\Delta)$. The main idea of our proof is as follows. Let μ be the minimum distance between any pair of points in P. Let $|pq|$ denote the Euclidean distance between two points $p, q \in P$. Since P has spread Δ, we have $|pq| \leqslant \mu\Delta$. For the matching M we define its *weight*, $w(M)$, to be the total length of its edges. Since M has $n/2$ edges,

$$w(M) = \sum_{pq \in M} |pq| = O(n\mu\Delta). \tag{1}$$

Recall that a pair of crossing segments can be flipped in two different ways as depicted in Figure 1(a). In the remainder of this section we show that one of these two flip operations



(a) (b)

Figure 2 Illustrations of the proofs of (a) Lemma 7 and (b) Lemma 6.

reduces $w(M)$ by at least $t\mu$, for some constant $t > 0$. Combining this with Equality (1) implies the existence of a flip sequence of length $O(n\Delta)$ that uncrosses M.

Take any two crossing edges pq and rs in M, and let o be their intersection point. We flip pq and rs to ps and rq, if $\angle roq \leqslant \pi/2$, and to pr and qs, otherwise. In other words, we flip pq and rs to the two edges that face the two smaller angles at o. In Lemma 7 we prove that this flip reduces the length of edges by at least $t\mu'$, for some constant $t > 0$, where μ' is the minimum distance between any pair of points in $\{p, q, r, s\}$. Since the minimum distance between pairs in $\{p, q, r, s\}$ is at least the minimum distance between pairs in P, our result follows. We use the following two lemmas in the proof of Lemma 7; we prove these two lemmas later.

▶ **Lemma 5.** *Let ab and cd be two crossing segments, and let o be their intersection point. Let μ'' be the minimum distance between any pair of points in $\{a, b, c, d\}$. If $\angle cob \leqslant \pi/3$, then*

$$(|ab| + |cd|) - (|ad| + |cb|) \geqslant \mu''.$$

▶ **Lemma 6.** *Let ab and cd be two perpendicular segments that cross each other. Let μ'' be the minimum distance between any pair of points in $\{a, b, c, d\}$. Then for any constant $t' \leqslant (2 - \sqrt{2})/2$ it holds that*

$$(|ab| + |cd|) - (|ad| + |cb|) \geqslant t'\mu''.$$

▶ **Lemma 7.** *Let pq and rs be two crossing segments, and let o be their intersection point. Let μ' be the minimum distance between any pair of points in $\{p, q, r, s\}$. If $\angle roq \leqslant \pi/2$, then for some constant t it holds that*

$$(|pq| + |rs|) - (|ps| + |rq|) \geqslant t\mu'.$$

Proof. If $\angle roq < \pi/3$, then our claim follows, with $t = 1$, from Lemma 5 where p, q, r, s play the roles of a, b, c, d, respectively. Assume that $\angle roq \geqslant \pi/3$. Observe that $\angle roq = \angle pos$.

After a suitable rotation and/or a horizontal reflection and/or relabeling assume that $|pq| \geqslant |rs|$, pq is horizontal, p is to the left of q, and r lies above pq. Rotate rs counterclockwise about o, while keeping o on this segment, until rs is vertical. See Figure 2(a). After this rotation, let r' and s' denote the two points that correspond to r and s, respectively.

▶ **Claim 8.** *$|r'p| > |rp|/2$ and $|qs'| > |qs|/2$.*

We prove only the first inequality of this claim; the proof of the second inequality is analogous. Since $r'p$ is the hypotenuse of the right triangle $\triangle r'op$, we have $|r'o| \leqslant |r'p|$.

Since $\triangle r'or$ is isosceles and $\angle r'or \leqslant \pi/6$, we have $|rr'| < |r'o|$, and thus, $|rr'| < |r'p|$. By the triangle inequality we have $|rp| \leqslant |rr'| + |r'p| < 2|r'p|$, which implies $|r'p| > |rp|/2$. This proves Claim 8.

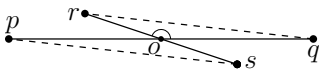
Observe that $|r'q| \geqslant |rq|$, $|ps'| \geqslant |ps|$, $|r's'| = |rs|$, and by Claim 8, $|r'p| \geqslant |rp|/2$ and $|qs'| \geqslant |qs|/2$. Thus, the minimum distance μ'' between any pair of points in $\{p, q, r', s'\}$ is not smaller than half the minimum distance between any pair of points in $\{p, q, r, s\}$, i.e., $\mu'' \geqslant \mu'/2$. Lemma 6 implies that $(|pq| + |r's'|) - (|ps'| + |r'q|) \geqslant t'\mu''$, for some constant $t' > 0$, where p, q, r', s' play the roles of a, b, c, d, respectively. We will see in the proof of Lemma 6 that this inequality is valid for any positive constant $t' \leqslant (2 - \sqrt{2})/2$. Combining these inequalities, we get

$$\begin{aligned}(|pq| + |rs|) - (|ps| + |rq|) &\geqslant (|pq| + |r's'|) - (|ps'| + |r'q|) \\ &\geqslant \frac{2-\sqrt{2}}{2}\mu'' \geqslant \frac{2-\sqrt{2}}{4}\mu'.\end{aligned}$$

Therefore, the claimed inequality in the statement of this lemma is valid for any positive constant $t \leqslant (2 - \sqrt{2})/4$. ◀

▶ **Note 9.** *The constants $t' = (2 - \sqrt{2})/2$ and $t = (2 - \sqrt{2})/4$ in the proofs of Lemmas 6 and 7 are not optimized. To keep our proofs short and simple, we avoid optimizing these constants.*

▶ **Note 10.** *The angle constraint in the statement of Lemma 7 cannot be dropped; the figure to the right shows two crossing segments pq and rs for which $(|pq| + |rs|) - (|ps| + |rq|)$ tends to zero as $\angle roq$ tends to π.*



Proof of Lemma 5. We recall the simple fact that the largest side of every triangle always faces the largest angle of the triangle. Since $\angle cob \leqslant \pi/3$, we have that $\angle cbo \geqslant \pi/3$ or $\angle bco \geqslant \pi/3$. Without loss of generality assume that $\angle bco \geqslant \pi/3$, and thus, $\angle bco \geqslant \angle cob$. This implies that $|ob| \geqslant |cb|$. By a similar reasoning, we get that $|oa| \geqslant |ad|$ or $|od| \geqslant |ad|$. If $|oa| \geqslant |ad|$, then

$$|ab| + |cd| - (|ad| + |cb|) = (|oa| + |ob|) + |cd| - (|ad| + |cb|) \geqslant |cd| \geqslant \mu'',$$

and if $|od| \geqslant |ad|$, then

$$|ab| + |cd| - (|ad| + |cb|) = (|oa| + |ob|) + (|oc| + |od|) - (|ad| + |cb|) \geqslant |oa| + |oc| \geqslant |ac| \geqslant \mu''.$$ ◀

Proof of Lemma 6. Refer to Figure 2(b) for an illustration of the proof. Let o be the intersection point of ab and cd. Let o' be the intersection point between cb and the line that is perpendicular to cb. Without loss of generality assume that ob is longer than oc, i.e., $|ob| \geqslant |oc|$. Then $\angle ocb \geqslant \angle obc$, and thus, $\angle ocb \geqslant \pi/4$. Since $\angle oo'c = \pi/2$ and $\angle oco' = \angle ocb \geqslant \pi/4$, we get that $\angle coo'$ is the smallest angle in the triangle $\triangle oco'$, and thus, $o'c$ is its smallest side. By doing some simple algebra we get that $|o'c| \leqslant |oc|/\sqrt{2}$.

Let o'' be the intersection point between ad and the line that is perpendicular to ad. We consider two cases depending on which of oa and od is longer.

- $|oa| > |od|$: By a similar reasoning as for ob and oc we get that $|o''d| \leqslant |od|/\sqrt{2}$. Observe that $|ob| > |o'b|$ and $|oa| > |o''a|$. By combining these inequalities we get

$$\begin{aligned}(|ab| + |cd|) - (|ad| + |cb|) &= (|oa| + |ob|) + (|oc| + |od|) - (|o''a| + |o''d|) - (|o'c| + |o'b|)\\ &> |oc| + |od| - |o''d| - |o'c| \geqslant |oc| + |od| - \frac{|od|}{\sqrt{2}} - \frac{|oc|}{\sqrt{2}}\\ &= \left(1 - \frac{1}{\sqrt{2}}\right)(|oc| + |od|) = \frac{2-\sqrt{2}}{2}|cd| \geqslant \frac{2-\sqrt{2}}{2}\mu''.\end{aligned}$$

- $|oa| \leqslant |od|$: Again, by a similar reasoning as for ob and oc we get that $|o''a| \leqslant |oa|/\sqrt{2}$. Also, by a similar reasoning as in the previous case we get

$$\begin{aligned}(|ab| + |cd|) - (|ad| + |cb|) &\geqslant |oc| + |oa| - \frac{|oa|}{\sqrt{2}} - \frac{|oc|}{\sqrt{2}}\\ &= \frac{2-\sqrt{2}}{2}|ca| \geqslant \frac{2-\sqrt{2}}{2}\mu''.\end{aligned}$$

Therefore, the claimed inequality in the statement of this lemma is valid for any positive constant $t' \leqslant (2-\sqrt{2})/2$. ◀

3 Points in Convex Position

In this section we study the problem of uncrossing perfect matchings and spanning trees on points in convex position. For perfect matchings, Bonnet and Miltzow [7] exhibited two perfect matchings M_1 and M_2 on n points in the plane in convex position such that $f(M_1) \geqslant n/2 - 1$ and $F(M_2) \geqslant \binom{n/2}{2}$. The following theorem provides matching upper bounds for $f(\cdot)$ and $F(\cdot)$.

▶ **Theorem 11.** *For every perfect matching M on a set of n points in the plane in convex position we have $f(M) \leqslant \frac{n}{2} - 1$ and $F(M) \leqslant \binom{n/2}{2}$.*

Proof. The matching M contains $n/2$ edges. First we prove that $F(M) \leqslant \binom{n/2}{2}$. Notice that the number of crossings between the edges of M is at most $\binom{n/2}{2}$. We show that any flip reduces this number by at least one, and thus, our claim follows. Take any pair ab and cd of crossing edges of M. Flip this crossing, and let ac and bd be the new edges, after a suitable relabeling. After this flip operation, the crossing between ab and cd disappears. Moreover, any edge of M that crosses ac (or bd) used to cross ab or cd, and any edge of M that crosses both ac and bd used to cross both ab and cd. Therefore, the total number of crossings reduces by at least one, and thus, our claim follows.

Now, we prove, by induction on n, that $f(M) \leqslant n/2 - 1$. If $n = 2$, then M has only one edge, and thus, $f(M) = 0$. Assume that $n \geqslant 4$. First, we show how to transform M, by at most one flip, to a perfect matching M' containing a boundary edge, i.e., an edge of the boundary of the convex hull. Let $p_1, \ldots, p_n$ be the points in clockwise order. Let p_ip_j be an edge of M with minimum depth m. If $m = 0$, then $M' = M$ is a matching in which p_ip_j is a boundary edge. Suppose that $m \geqslant 1$. Without loss of generality assume that $i = 1$ and $j = m + 2$. Let p_k be the point that is matched to p_2 by M. Because of the minimality of m, the edge p_2p_k crosses p_1p_{m+2}. By flipping p_2p_k and p_1p_{m+2} to p_1p_2 and $p_{m+2}p_k$ we obtain M' in which p_1p_2 is a boundary edge. Let M'' be the matching on $n - 2$ points obtaining from M' by removing a boundary edge. By the induction hypothesis, it holds that

$$f(M) \leqslant 1 + f(M'') \leqslant 1 + \left(\frac{n-2}{2} - 1\right) = \frac{n}{2} - 1.$$ ◀

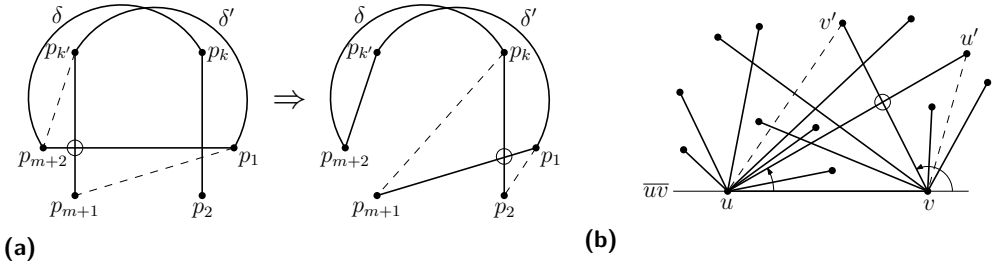


Figure 3 (a) Illustration of the proof of Lemma 12: Flipping p_1p_{m+2} and $p_{m+1}p_{k'}$ to p_1p_{m+1} and $p_{m+2}p_{k'}$, and then flipping p_1p_{m+1} and p_2p_k to p_1p_2 and p_kp_{m+1}. (b) Illustration of the proof of Lemma 13: vv' is the first counterclockwise edge incident on v that is crossed by some edges incident on u, and uu' is the first counterclockwise edge incident on u that crosses vv'.

In the rest of this section we study spanning trees. The argument of [21] for Hamiltonian cycles also extends to spanning trees, that is, if T_1 is a spanning tree on n points in the plane, then $F(T_1) = O(n^3)$. Also, by an argument similar to the one in the proof of Theorem 11, it can easily be shown that for every spanning tree T on n points in the plane in convex position we have that $F(T) = O(n^2)$. In this section we prove that $f(T) = O(n \log n)$. Recall that a boundary edge is an edge of the boundary of the convex hull.

▶ **Lemma 12.** *Any spanning tree on a point set in convex position can be transformed, by at most two flips, into a spanning tree containing a boundary edge.*

Proof. Let T be a spanning tree on n points in the plane in convex position, and let $p_1, \ldots, p_n$ be the points in clockwise order. Let p_ip_j be an edge of T with minimum depth m (recall the definition of depth from Section 1.3). If $m = 0$, then p_ip_j is a boundary edge. Suppose that $m \geqslant 1$. Without loss of generality assume that $i = 1$ and $j = m + 2$. Because of the minimality of m, all edges of T that are incident on $p_2, \ldots, p_{m+1}$ cross p_1p_{m+2}. We consider two cases with $m = 1$ and $m > 1$.

- $m = 1$. In this case $p_{m+1} = p_2$ and $p_{m+2} = p_3$. Let δ be the path between p_2 to p_3 in T, and let p_k be the vertex that is adjacent to p_2 in δ. If δ contains p_1, then we flip p_1p_3 and p_2p_k to p_1p_2 and p_3p_k; this gives a spanning tree in which p_1p_2 is a boundary edge. If δ does not contain p_1, then we flip p_1p_3 and p_2p_k to p_2p_3 and p_1p_k; this gives a spanning tree in which p_2p_3 is a boundary edge.
- $m > 1$. Let δ be the path between p_2 to p_{m+2} in T, and let p_k be the vertex that is adjacent to p_2 in δ. If δ contains p_1, then we flip p_1p_{m+2} and p_2p_k to p_1p_2 and $p_{m+2}p_k$; this gives the a spanning tree in which p_1p_2 is a boundary edge. Assume that δ does not contain p_1. Let δ' be the path between p_{m+1} to p_1 in T, and let $p_{k'}$ be the vertex that is adjacent to p_{m+1} in δ'; it may be that $k' = k$. If δ' contains p_{m+2}, then we flip p_1p_{m+2} and $p_{m+1}p_{k'}$ to $p_{m+1}p_{m+2}$ and $p_1p_{k'}$; this gives a spanning tree in which $p_{m+1}p_{m+2}$ is a boundary edge. Assume that δ' does not contain p_{m+2}. See Figure 3(a). In this case we have that $k' \neq k$, because otherwise T would have a cycle. First we flip p_1p_{m+2} and $p_{m+1}p_{k'}$ to p_1p_{m+1} and $p_{m+2}p_{k'}$, then we flip p_1p_{m+1} and p_2p_k to p_1p_2 and p_kp_{m+1}; this gives a spanning tree in which p_1p_2 is a boundary edge. ◀

For the following lemma we do not need the vertices to be in convex position.

▶ **Lemma 13.** *Let T be a spanning tree containing an edge uv such that every other edge is incident on either u or v. Then $f(T) \leqslant \min(\deg(u), \deg(v)) - 1$, and this bound is tight.*

Proof. After a suitable rotation and/or a horizontal reflection and/or relabeling assume that $\overline{uv}$ is horizontal, u is to the left of v, and that $\deg(v) \leqslant \deg(u)$. The edges that are incident on points above $\overline{uv}$ do not cross the edges incident on points below $\overline{uv}$. Thus, the crossings above $\overline{uv}$ can be handled independently of the ones below $\overline{uv}$. Because of symmetry, we describe how to handle the crossings above $\overline{uv}$. See Figure 3(b). We show how to increase, by one flip, the number of free edges that are incident on v. By repeating this process, our claim follows. To that end, let v' be the first vertex, in counterclockwise order, that is adjacent to v, and such that vv' is crossed by at least one edge incident on u. Let u' be the first vertex, in counterclockwise order, that is adjacent to u, and such that uu' crosses vv'; see Figure 3(b). Flip this crossing to obtain new edges vu' and uv'. The edge vu' is free, because otherwise uu' cannot be the first counterclockwise edge that crosses vv'. Moreover, any edge that is crossed by uv' used to be crossed by uu'. Thus, the number of free edges that are incident on v increases by at least one. By repeating this process, after at most $\deg(v) - 1$ iterations, all incident edges on v become free (notice that the edge uv is already free); this transforms T to a plane spanning tree. This proves the first statement of the lemma.

Recall that the statement of this lemma is not restricted to points in convex position, and thus, the vertices of our tight example do not need to be in convex position. To verify the tightness of the bound, consider a tree in which every edge incident on v (except uv) is crossed by exactly one of the edges incident on u, and every edge incident on u crosses at most one of the edges incident on v. This tree needs exactly $\deg(v) - 1$ flips to be transformed to a plane tree. ◀

▶ **Theorem 14.** *For every spanning tree T on n points in the plane in convex position we have that $f(T) = O(n \log n)$.*

Proof. We present a recursive algorithm that uncrosses T by $O(n \log n)$ flips. As for the base case, if $n \leqslant 3$, then T is plane, and thus, no flip is needed. Assume that $n \geqslant 4$. By Lemma 12, by at most two flips, we can transform T to a tree T' containing a boundary edge uv. Contract the edge uv and denote the resulting tree with $n - 1$ vertices by T''; this can be done by removing the vertex u together with its incident edges, and then connecting its neighbors, by straight-line edges, to v. We call every such new edge a *u-edge*. Recursively uncross T'' with $f(T'')$ flips. During the uncrossing process of T'', whenever we flip/remove a u-edge, we call the new edge that gets connected to v a u-edge. After uncrossing T'' we return the vertex u back and connect it to v. Then we remove every u-edge vv', which is incident on v, and connect v' to u. In the resulting tree, every crossing is between an edge that is incident on u and an edge that is incident on v. Thus, after at most $2 + f(T'')$ flips, T can be transformed into a tree in which any two crossing edges are incident on u and v. Then by Lemma 13, we can obtain a plane tree by performing at most $\min(\deg(u), \deg(v)) - 1$ more flips. Notice that the flip operation does not change the degree of vertices, and thus, every vertex in the resulting tree has the same degree as in T. Therefore, we have that

$$\begin{aligned} f(T) &\leqslant 2 + f(T'') + \min(\deg(u), \deg(v)) - 1 \\ &= 1 + \min(\deg(u), \deg(v)) + f(T''). \end{aligned}$$

It remains to show that $f(T) = O(n \log n)$. To that end, we interpret the above recursion by a union-find data structure with the linked-list representation and the weighted-union heuristic [12, Chapter 21]. The number of flips in the above recursion can be interpreted as the total time for union operations as follows: each time that we contract an edge uv and recurse on a smaller tree we perform at most $1 + \min(\deg(u), \deg(v))$ flips. Consider every vertex x of T as a set with $\deg(x)$ elements. Also, assume that all the elements of these sets are pairwise distinct. Thus, we have n disjoint sets of total size $2(n - 1)$; this is coming from the fact that T has $n - 1$ edges and its total vertex degree is $2(n - 1)$. The

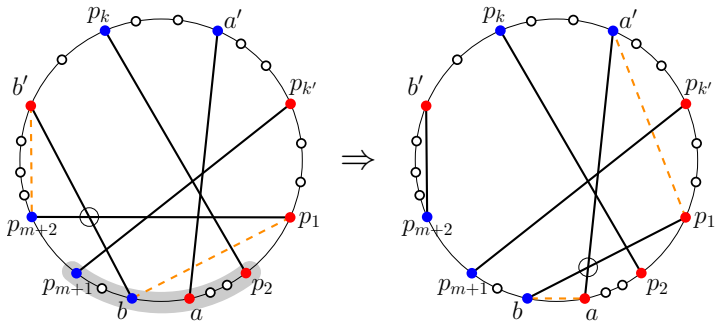


Figure 4 Illustration of the proof of Theorem 15. Flipping bb' and p_1p_{m+2} to $b'p_{m+2}$ and bp', and then flipping bp_1 and aa' to p_1a' and ab.

contraction of an edge uv can be interpreted as a union operation of the sets u and v whose cost (number of flips) is at most $1 + \min(|u|, |v|)$, where $|x|$ denotes the size of the set x. From the union-find data structure we have that the cost of a sequence of s operations on m elements is $O(s + m \log m)$. In our case, the number m of elements is $2(n-1)$, and the number s of union operations (edge contractions) is $n-3$ (no contraction is needed when we hit the base case). Thus, it follows that the total cost (the total number of flips) is $O(n \log n)$. ◀

4 Bichromatic Matchings

In this section we study the problem of uncrossing perfect bichromatic matchings for points in convex position and for semi-collinear points. Let $n \geqslant 2$ be an even integer, and let P be a set of n points in the plane, $n/2$ of which are colored red and $n/2$ are colored blue. If P is in general position, then for any bichromatic matching M on P, the best known upper bound for both $f(M)$ and $F(M)$ is the $O(n^3)$ bound that has been proved in [7, 21]. If P is in convex position, the $n/2 - 1$ and $\binom{n/2}{2}$ lower bounds that are shown in [7] for $f(\cdot)$ and $F(\cdot)$, respectively, in the monochromatic setting, also hold in the bichromatic setting. Theorem 11 implies that the $\binom{n/2}{2}$ bound for $F(\cdot)$ is tight. The following theorem gives an upper bound on $f(\cdot)$ for points in convex position.

▶ **Theorem 15.** *For every perfect bichromatic matching M on n points in the plane in convex position we have that $f(M) \leqslant n - 2$.*

Proof. Our proof is by induction on n. If $n = 2$, then $f(M) = 0$. Assume that $n \geqslant 4$. First we show how to transform M, by at most two flips, to a perfect bichromatic matching M' containing a boundary edge. Let $p_1, \ldots, p_n$ be the points in clockwise order. Let p_ip_j be an edge of M with minimum depth m. If $m = 0$, then $M' = M$ is a matching in which p_ip_j is a boundary edge. Suppose that $m \geqslant 1$. Without loss of generality assume that $i = 1$, $j = m + 2$, p_1 is red, and p_{m+2} is blue as in Figure 4. Let p_k and $p_{k'}$ be the points that are matched to p_2 and p_{m+1}, respectively; it may be that $m + 1 = 2$ and $k' = k$. Because of the minimality of m, all edges that are incident on points $p_2, \ldots, p_{m+1}$ cross p_1p_{m+2}. If p_2 is blue, then by flipping p_1p_{m+2} and p_2p_k to p_1p_2 and $p_{m+2}p_k$ we obtain M' in which p_1p_2 is a boundary edge. Assume that p_2 is red. If p_{m+1} is red, then by flipping p_1p_{m+2} and $p_{m+1}p_{k'}$ to $p_{m+1}p_{m+2}$ and $p_1p_{k'}$ we obtain M' in which $p_{m+1}p_{m+2}$ is a boundary edge. Assume that p_{m+1} is blue. See Figure 4. To this end, p_2 and p_{m+1} have different colors, and thus, $m + 1 \neq 2$ and $k' \neq k$.

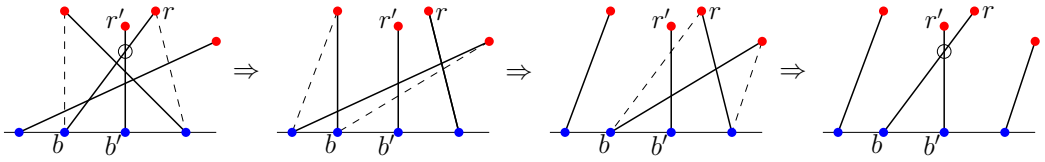


Figure 5 Reappearance of the crossing between br and $b'r'$.

For an illustration of the rest of the proof, follow Figure 4. The sequence $p_2, \ldots, p_{m+1}$ starts with a red point and ends with a blue point. Thus, in this sequence there are two points of distinct colors, say a and b, that are consecutive. Let b be the first blue point after p_1. Let a' and b' be the two points that are matched to a and b respectively. By flipping bb' and p_1p_{m+2} to $b'p_{m+2}$ and bp_1, and then flipping bp_1 and aa' to p_1a' and ab we obtain M' in which ab is a boundary edge.

Let M'' be the bichromatic matching on $n-2$ points obtaining from M' by removing a boundary edge. By the induction hypothesis, it holds that

$$f(M) \leqslant 2 + f(M'') \leqslant 2 + ((n-2)-2) = n-2.$$ ◀

In the rest of this section we study the case where P is semi-collinear, i.e., its blue points are on a straight line and its red points are in general position. Semi-collinear points have been studied in may problems related to plane matchings (see e.g., [5, 6, 10]). We prove that for every perfect bichromatic matching M on P, it holds that $f(M) = O(n^2)$. Before we prove this upper bound, observe that similar to the general position setting, in the semi-collinear setting the total number of crossings might increase after a flip. Also, it is possible that a crossing, that has disappeared after a flip, reappears after some more flips (see the crossing between br and $b'r'$ in Figure 5). The $O(n^2)$ upper bound given in [7] for $f(\cdot)$ on uncolored points, which is obtained by connecting the two leftmost points of a crossing, does not apply to our semi-collinear bichromatic setting, because in this setting the two leftmost points might have the same color. These observations imply that there is no straightforward way of getting a good upper bound.

Let ℓ be the line that contains all the blue points of P. By a suitable rotation we assume that ℓ is horizontal. For every perfect bichromatic matching M on P, the edges of M, that are above ℓ, do not cross the ones that are below ℓ. Thus, we can handle these two sets of edges independently of each other. Therefore, in the rest of this section we assume that the red points of P lie above ℓ. Recall that P contains $n/2$ blue points and $n/2$ red points.

▶ **Lemma 16.** *Let M be a perfect bichromatic matching on P in which the rightmost blue point b is matched to the topmost red point r. If $M \setminus \{br\}$ is plane, then $f(M) \leqslant \frac{n}{2} - 1$, and this bound is tight.*

Proof. See Figure 6(a) for an illustration of the statement of this lemma; notice that if we remove br from M, then we get a plane matching. Our proof is by induction on n. If $n = 2$, then M has one edge which is plane, and thus, $f(M) = 0$. Assume that $n \geqslant 4$. If br does not intersect any other edge, then M is plane and $f(M) = 0$. Suppose that br intersects some edges of $M \setminus \{br\}$, and let R' be the set of the red endpoints of those edges; see Figure 6(a). Let r' be the first red point in the counterclockwise order of the red points around b; observe that r' belongs to R'. Let b' be the blue point that is matched to r'. Flip br and $b'r'$ to br' and $b'r$ as in Figure 6(b), and let M' be the resulting matching. The edge br' does not cross any other edge of M', because of our choice of r', but the edge $b'r$ may cross some edges of M'. Let M'' be the subset of edges of M' that are to the left of $\overline{b'r'}$; see Figure 6(b). Notice that $br' \notin M''$, and thus M'' is a matching on at most $n-2$ points.

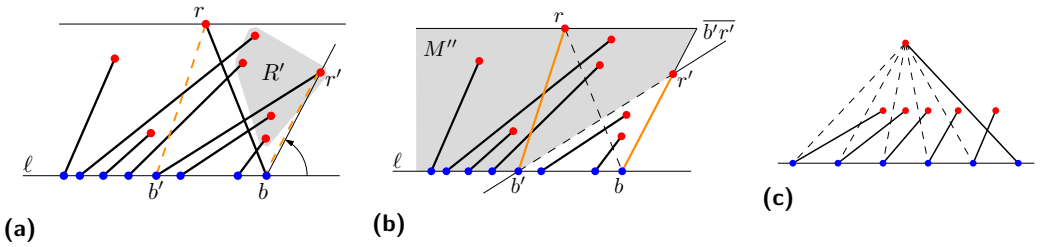


Figure 6 Illustration of the proof of Lemma 16.

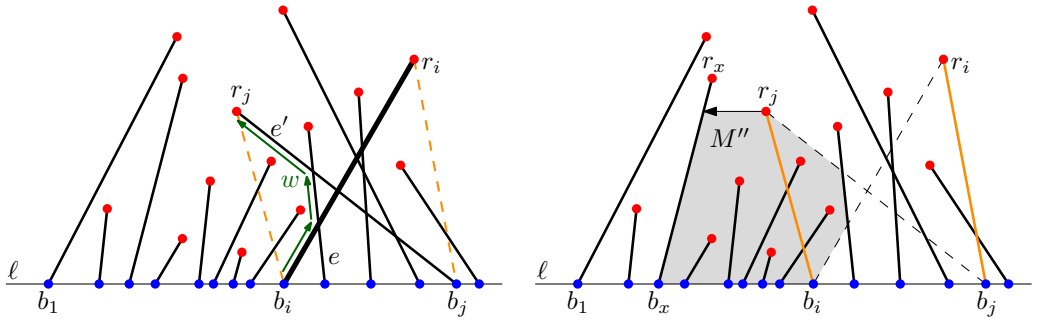


Figure 7 Illustration of the proof of Theorem 17.

Because of the planarity of $M \setminus \{br\}$ and since r is the topmost red point, we have that $M' \setminus M''$ is plane. Moreover, M'' and $M' \setminus M''$ are separated by $\overline{b'r'}$. Observe that b' is the rightmost blue point in M'' that is matched to the topmost red point r, moreover, $M'' \setminus \{b'r\}$ is plane. Therefore, we can repeat the above process on M'', which is a smaller instance of the initial problem. By the induction hypothesis, it holds that

$$f(M) = 1 + f(M'') \leqslant 1 + \left(\frac{n-2}{2} - 1\right) = \frac{n}{2} - 1.$$

To verify the tightness, Figure 6(c) shows a matching example for which we need exactly $n/2-1$ flips to transform it to a plane matching. Each time there exists exactly one crossing, and after flipping that crossing, only one other crossing appears (except for the last flip). ◀

▶ **Theorem 17.** *For every perfect bichromatic matching M on P we have $f(M) \leqslant \frac{n^2}{8} + \frac{n}{4}$.*

Proof. We present an iterative algorithm that uncrosses M by $O(n^2)$ flips. Let $b_1, \ldots, b_{n/2}$ be the blue points from left to right. By a suitable relabeling assume that $M = \{b_1r_1, \ldots, b_{n/2}r_{n/2}\}$. To simplify the description of the proof, we add, to M, a dummy edge b_0r_0 such that b_0 is a blue point on ℓ that is to the left of all the blue points, r_0 is a red point that is higher than all the red points, and all points of P are to the right of $\overline{b_0r_0}$.

We describe one iteration of our algorithm. If M is plane, then the algorithm terminates. Assume that M is not plane. Let $i \in \{1, \ldots, n/2\}$ be the smallest index such that b_ir_i intersects some edges of M; see Figure 7-left. To simplify the rest of our description, we refer to the current iteration as iteration i. Notice that the blue endpoint of every non-free edge is strictly to the right of b_{i-1}. Let r_j be the first red point that we meet in the following walk along the edges of M. Starting from b_i, we walk along b_ir_i until we see the first edge e that crosses b_ir_i. Then we turn left on e and keep walking until we see a red point or another crossing edge. If we see a red point, then we call it r_j and finish the walk. If we see a crossing edge e', then we turn left on e' and keep walking until we see a red point, namely r_j, or we see another crossing edge. In the latter case we repeat this process and stop as soon as we

see the first red point, which we call it r_j. Let b_j be the blue point that is matched to r_j. Let w denote the convex polygonal path that we traversed from b_i to r_j.

Flip b_ir_i and b_jr_j to b_ir_j and b_jr_i, and let M' denote the resulting matching. See Figure 7-right. Shoot a horizontal ray, from r_j, to the left, and stop as soon as it hits an edge b_xr_x in M'. Let M'' be the subset of the edges of M' that are incident on $b_{x+1}, \ldots, b_i$, that is, $M'' = \{b_{x+1}r_{x+1}, \ldots, b_{i-1}r_{i-1}, b_ir_j\}$. By the way that we picked r_j, the edges of M'' are in a convex region whose interior is disjoint from the edges of $M' \setminus M''$; this convex region is bounded by ℓ, b_xr_x, w, and the ray from r_j, as depicted in Figure 7-right. The matching M'' has $i - x$ edges. Observe that, in M'', we have that b_i is the rightmost blue point that is matched to the topmost red point r_j, and $M'' \setminus \{b_ir_j\}$ is plane. Thus, by Lemma 16 we can uncross M'' by at most $i - x - 1$ flips. To this end, we have transformed M to a matching in which the edges that are incident on $b_1, \ldots, b_i$ are free. The total number of flips performed in iteration i is at most $1 + (i - x - 1) = i - x \leqslant i$.

In the next iteration, the smallest index i', for which $b_{i'}r_{i'}$ is not free, is larger than i. Thus, this smallest index moves at least one step to the right after each iteration. This means that the number of free edges, that are connected to the blue points of lower indices, increases. Therefore, after at most $n/2$ iterations our algorithm terminates. The total number of flips is

$$f(M) \leqslant \sum_{i=1}^{n/2} i = \frac{n^2}{8} + \frac{n}{4}.$$ ◀

5 Conclusions

We investigated the number of flips that are necessary and sufficient to reach a non-crossing perfect matching on n points in the plane. It is known that the minimum and the maximum lengths of a flip sequence are $O(n^2)$ and $O(n^3)$, respectively. We proved, with a new approach, that the minimum length of a flip sequence is $O(n\Delta)$ where Δ is the spread of the points set; this improves the bound for point sets with sublinear spread. A natural open problem is to improve any of these bounds. Another open problem is to improve our $O(n \log n)$ upper bound on the number of sufficient flips to reach a plane spanning tree on points in convex position, or to show that this bound is tight. One potential way to do this, is that in Theorem 14, we get a boundary edge uv such that one of u or v has a constant degree.

It is worth mentioning that the number of flips, in a flip sequence, is highly dependent on the *order* in which we choose crossings to flip, and the *type* of a flip that we perform (among the two possible types). This dependency can be used to improve the bounds on the minimum number of flips. In Theorems 11, 14, 15, and 17 we used the order and proved some upper bounds, while in Theorem 4 we used the flip type. One may think of using the order and the flip type together to improve the current bounds. Notice that for bichromatic matchings, spanning trees, and Hamiltonian cycles only one type of flip is possible, and thus, only the order can be used for further improvements. Also, notice that none of the order and the flip type can be used to improve the bounds on the maximum number of flips, because, in this case, an adversary chooses the order and the type.

References

1 Oswin Aichholzer, Andrei Asinowski, and Tillmann Miltzow. Disjoint compatibility graph of non-crossing matchings of points in convex position. *Electr. J. Comb.*, 22(1):P1.65, 2015.

2 Oswin Aichholzer, Sergey Bereg, Adrian Dumitrescu, Alfredo García Olaverri, Clemens Huemer, Ferran Hurtado, Mikio Kano, Alberto Márquez, David Rappaport, Shakhar

Smorodinsky, Diane L. Souvaine, Jorge Urrutia, and David R. Wood. Compatible geometric matchings. *Comput. Geom.*, 42(6-7):617–626, 2009.

3 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015.

4 Greg Aloupis, Luis Barba, Stefan Langerman, and Diane L. Souvaine. Bichromatic compatible matchings. *Comput. Geom.*, 48(8):622–633, 2015.

5 Greg Aloupis, Jean Cardinal, Sébastien Collette, Erik D. Demaine, Martin L. Demaine, Muriel Dulieu, Ruy Fabila Monroy, Vi Hart, Ferran Hurtado, Stefan Langerman, Maria Saumell, Carlos Seara, and Perouz Taslakian. Non-crossing matchings of points with geometric objects. *Computational Geometry: theory and Applications*, 46(1):78–92, 2013.

6 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Bottleneck bichromatic plane matching of points. In *Proceedings of the 26th Canadian Conference on Computational Geometry* (*CCCG*), pages 431–435, 2014.

7 Édouard Bonnet and Tillmann Miltzow. Flip distance to a non-crossing perfect matching. *EuroCG*, 2016.

8 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Comput. Geom.*, 42(1):60–80, 2009.

9 Prosenjit Bose and Sander Verdonschot. A history of flips in combinatorial triangulations. In *XIV Spanish Meeting on Computational Geometry* (*EGC*), pages 29–44, 2011.

10 John Gunnar Carlsson, Benjamin Armbruster, Saladi Rahul, and Haritha Bellam. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applcations*, 25(4):245–262, 2015.

11 Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 21: Data structures for Disjoint Sets. The MIT Press and McGraw-Hill Book Company, second edition, 2001.

13 Herbert Edelsbrunner, Pavel Valtr, and Emo Welzl. Cutting dense point sets in half. *Discrete & Computational Geometry*, 17(3):243–255, 1997.

14 Ferran Hurtado, Marc Noy, and Jorge Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

15 Mashhood Ishaque, Diane L. Souvaine, and Csaba D. Tóth. Disjoint compatible geometric matchings. *Discrete & Computational Geometry*, 49(1):89–131, 2013.

16 Charles L Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

17 Yoshiaki Oda and Mamoru Watanabe. The number of flips required to obtain non-crossing convex cycles. In *Proceedings of the International Conference on Computational Geometry and Graph Theory* (*KyotoCGGT*), pages 155–165, 2007.

18 Alfredo García Olaverri, Clemens Huemer, Ferran Hurtado, and Javier Tejel. Compatible spanning trees. *Comput. Geom.*, 47(5):563–584, 2014.

19 Pavel Valtr. *Planar point sets with bounded ratios of distances.* PhD thesis, Fachbereich Mathematik, Freie Universität Berlin, 1994.

20 Pavel Valtr. Lines, line-point incidences and crossing families in dense sets. *Combinatorica*, 16(2):269–294, 1996.

21 Jan van Leeuwen and Anneke A. Schoone. Untangling a travelling salesman tour in the plane. In *Proceedings of the 7th Conference Graphtheoretic Concepts in Computer Science* (*WG*), pages 87–98, 1981.

Boundary Labeling for Rectangular Diagrams

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada
jit@scs.carleton.ca

Paz Carmi
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel
carmip@cs.bgu.ac.il

J. Mark Keil
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
mark.keil@usask.ca

Saeed Mehrabi
School of Computer Science, Carleton University, Ottawa, Canada
saeed.mehrabi@carleton.ca

Debajyoti Mondal
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
d.mondal@usask.ca

Abstract

Given a set of n points (sites) inside a rectangle R and n points (label locations or ports) on its boundary, a boundary labeling problem seeks ways of connecting every site to a distinct port while achieving different labeling aesthetics. We examine the scenario when the connecting lines (leaders) are drawn as axis-aligned polylines with few bends, every leader lies strictly inside R, no two leaders cross, and the sum of the lengths of all the leaders is minimized. In a k-sided boundary labeling problem, where $1 \leq k \leq 4$, the label locations are located on the k consecutive sides of R.

In this paper we develop an $O(n^3 \log n)$-time algorithm for 2-sided boundary labeling, where the leaders are restricted to have one bend. This improves the previously best known $O(n^8 \log n)$-time algorithm of Kindermann et al. (Algorithmica, 76(1):225–258, 2016). We show the problem is polynomial-time solvable in more general settings such as when the ports are located on more than two sides of R, in the presence of obstacles, and even when the objective is to minimize the total number of bends. Our results improve the previous algorithms on boundary labeling with obstacles, as well as provide the first polynomial-time algorithms for minimizing the total leader length and number of bends for 3- and 4-sided boundary labeling. These results settle a number of open questions on the boundary labeling problems (Wolff, Handbook of Graph Drawing, Chapter 23, Table 23.1, 2014).

2012 ACM Subject Classification Theory of computation, Theory of computation → Algorithm design techniques, Theory of computation → Computational geometry

Keywords and phrases Boundary labeling, Dynamic programming, Outerstring graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.12

Related Version See [8], `https://arxiv.org/abs/1803.10812` for the full version of the paper.

Funding Research of Prosenjit Bose and Saeed Mehrabi is supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC). Saeed Mehrabi is also supported by a Carleton-Fields postdoctoral fellowship. Debajyoti Mondal is supported in part by Global Water Futures project (GWF) and Natural Sciences and Engineering Research Council of Canada (NSERC).

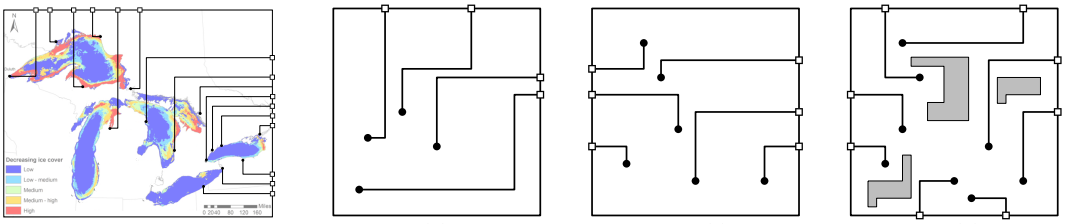


Figure 1 (a) A 1-bend 2-sided boundary labeling (i.e., with *po*-leaders) on a geographic map showing (ice cover on the Great Lakes [14]). (b) A 2-bend 2-sided boundary labeling (i.e., with *opo*-leaders). This example does not have a feasible solution with 1-bend leaders. (c) Boundary labeling in 1-bend opposite 2-sided model. (d) A 1-bend 4-sided boundary labeling in the presence of obstacles.

1 Introduction

Labeling problems appear in a variety of scenarios such as in annotating educational diagrams, wiring schematics, system manuals, as well as in many information visualization and engineering applications. The increasing trend of automation in these areas has motivated the research in labeling algorithms. Crossings among the *leaders* (i.e., the lines connecting labels to the sites), number of bends per leader, and the sum of leader lengths are some important aesthetics of a diagram labeling. To achieve clarity and better readability, all these parameters are often preferred to be kept small.

Many labeling problems are NP-hard [12, 5]. A rich body of research attempts to develop efficient approximation and heuristic algorithms [13, 15, 10, 21, 22], both in the static and the dynamic settings [3, 10]. In this paper we examine a well-known variant of the labeling problem called *b-bend k-sided boundary labeling*, e.g., see Figure 1. The input for this problem is a set of kn sites and kn ports, where the sites lie in the interior of a rectangle R, the ports are located on k consecutive sides of R, and each side contains n ports. Both the sites and ports are represented as points. The goal is to decide whether each site can be connected to a unique port using axis-aligned leaders such that the leaders are disjoint, each leader lies strictly inside R and each leader has at most b bends. If such a labeling exists, then we compute a labeling that optimizes these labeling aesthetics. We examine two such optimization criteria: one is to minimize the sum of the leader lengths, and the other is to minimize the total number of bends.

The strict-containment inside R, bend restrictions and orthogonal constraints impose certain shapes on the leader. An orthogonal leader containing exactly one bend (resp., two bends) is known as a *po-leader* (resp., an *opo-leader*)[1] [17]. We note that there are 1-bend leaders with 135° degrees at the bend, which are knowns as *do-leaders* [2]. Since we are only interested in orthogonal leaders in this paper, we say 1-bend leaders to always mean "*po*-leaders" for the rest of the paper.

Related work. Boundary labeling has been an active area of research in the last decade, e.g., see the surveys [1, 20]. The boundary labeling problem was first introduced by Bekos et al. [6]. They gave $O(n \log n)$-time algorithms to decide labeling feasibility for the 1-bend

[1] The letters 'o' and 'p' stand for 'orthogonal' and 'parallel', respectively. So, an *opo*-leader starts orthogonally at the site, and ends orthogonally at the port.

1-sided and opposite 2-sided models, i.e., the labels are located on two opposite sides of R. In addition, they gave an $O(n^2)$-time algorithm that minimizes the total leader length. For the 2-bend 4-sided model, they could test the feasibility in $O(n \log n)$ time and reduced the length minimization to a minimum-cost bipartite matching problem. Benkert et al. [7] improved Bekos et al.'s [6] result on the 1-bend 1-sided model by devising an $O(n \log n)$-time algorithm for the length minimization. They also considered general cost functions (i.e., beyond Euclidean length), as well as other types of leaders. We refer the reader to [19, 4] for other variants of boundary labeling problem.

The 2-sided model considered by Bekos et al. [6] and Benkert et al. [7] is an opposite-sided model, i.e., ports are placed on two opposite sides of R. This model is different from the adjacent 2-sided model, where the labels are always placed on adjacent sides. The adjacent 2-sided model was first considered by Kindermann et al. [17]. For the 1-bend 2-sided model, they gave an $O(n^2)$-time algorithm to check feasibility, and an $O(n^8 \log n)$-time algorithm for total leader length minimization; to our knowledge, this is the fastest algorithm known for the 1-bend 2-sided model. Note that the labeling problem in this model seems surprisingly more difficult than the corresponding opposite 2-sided model (also mentioned by Kindermann et al. [17]). For the 1-bend 3-sided (resp., 4-sided) model, they gave an $O(n^4)$-time (resp., $O(n^9)$-time) algorithm for checking the labeling feasibility, but they were unable to solve the length minimization problem. They posed this as an open question, i.e., can a minimum-length solution for the 3- and 4-sided boundary labeling be computed in polynomial time? These challenges motivated us to examine the adjacent model in more detail.

Fink and Suri [11] studied the boundary labeling problem in the presence of *obstacles*. In addition to the set of sites, they allowed a set of orthogonal polygons (equivalently, obstacles) to lie inside R. The objective is to minimize the total leader length with the constraint that the leaders must not intersect the obstacles. They gave polynomial-time algorithms for minimizing the total leader length in the 1-sided and opposite 2-sided models, but the running time of these algorithms while using *po*- and *opo*-leaders is fairly high, i.e., $O(n^4)$, $O(n^8)$ for the 1-sided model, and $O(n^9)$, $O(n^{21})$ for the opposite 2-sided model. They also examined the case when the leaders have non-uniform lengths and the leader locations can be chosen, which they proved to be NP-hard.

A different generalization of boundary labeling considers sliding ports, i.e., labels are assigned disjoint intervals on the boundary of R, and a site can be connected to any point in such an interval. In the 1-sided model, Bekos et al. [6] gave an $O(n^2)$-time algorithm that can minimize the total number of bends using *opo*-leader (they did not require the *opo*-leaders to lie strictly inside R). They posed an open question to determine the time complexity for the 3- and 4-sided case. Benkert et al. [7] considered bend minimization with *po*-leaders. They gave an $O(n^2)$-time algorithm for the 1-sided model, and $O(n^8)$-time algorithm for the opposite 2-sided model. The 'Handbook of Graph Drawing' [20] lists a number of open problems related to the minimization of the total number of bends for different variants of boundary labeling.

The 1-, 3- and 4-sided models for the boundary labeling problem are always adjacent models, but a 2-sided model can be either adjacent or opposite. Throughout the paper we will refer to the 'opposite' variant as an 'opposite 2-sided' model.

Our contributions. We give an algorithm for the 1-bend 2-sided boundary labeling problem that minimizes the total leader length in $O(n^3 \log n)$ time (if such a labeling exists). Ours is an adjacent model and uses *po*-leaders, and hence improves the $O(n^8 \log n)$-time algorithm of Kindermann et al. [17]. Since the best known algorithm for the length minimization in the

1-bend opposite 2-sided model takes $O(n^2)$ time [7], our result raises an intriguing question that whether the adjacent boundary labeling model can further be improved to reach (or, even break) the $O(n^2)$ barrier.

We show that many variants of the boundary labeling problems can be related to outerstring graphs, where the minimization of total leader lengths or bends reduces to an optimization problem in those outerstring graphs. We notice that this relation is previously pointed out in a different context [18]. This idea leads us to the following results:

- The first polynomial-time algorithm with a running time of $O(n^6)$ for the 1-bend 3-sided and 4-sided boundary labeling problem that minimize the total leader length. This settles the time-complexity question posed by Kindermann et al. [17].
- Polynomial-time algorithms for minimizing the total leader length or the total number of bends, even in the presence of obstacles. Our algorithms work for both *po*- and *opo*-leaders, as well as for all possible distributions of the ports to the boundary of R, i.e., both adjacent and opposite models. The running time for the opposite 2-sided model is $O(n^6)$ for *po*-leaders and $O(n^9)$ for *opo*-leaders; these improve, respectively, the $O(n^9)$- and $O(n^{21})$-time algorithms of Fink and Suri [11]. This technique can also be applied to the sliding port model, which settles the time-complexity question posed in [6, 20] related to the bend minimization.

2 Computing 1-Bend 2-Sided Boundary Labelings

In this section we give an $O(n^3 \log n)$-time algorithm to find a solution to the 1-bend 2-sided boundary labeling problem. Throughout this section, we assume that the sites and ports are in general position, i.e., no axis-aligned straight line passing through a site intersects a port or another site. Consequently, each leader must have exactly one bend. We thus omit the term '1-bend' in the rest of this section. Moreover, we assume that the ports lie on the top and right sides of the rectangle R.

2.1 Technical Background

Let $R(t), R(b), R(l), R(r)$ be the *top, bottom, left and right sides* of R, respectively. An *xy-separating curve* is an axis-aligned xy-monotone polygonal chain that starts at the bottom-left corner of R and ends at the top-right corner of R. A 2-sided boundary labeling solution is *xy-separated* if there exists an xy-separating curve such that the leaders incident to $R(t)$ (resp., $R(r)$) lie on or above (resp., below) the xy-separating curve.

▶ **Lemma 1** (Kindermann et al. [17]). *If a 2-sided boundary labeling problem has an affirmative solution with 1-bend leaders, then there exists such an xy-separated solution that minimizes the sum of all leader lengths.*

Figure 2(a) illustrates an xy-separated solution of a 2-sided boundary labeling problem. An xy-separated curve is shown in a light-green. Let $\mathcal{I}$ be an instance of a 2-sided boundary labeling problem. Without loss of generality assume that the ports are distributed along the sides $R(t)$ and $R(r)$. Let $ports(R(t))$ (resp., $ports(R(r))$) be the set of ports along $R(t)$ (resp., $R(r)$). A leader is called *inward* if the 90° angle formed at its bend point contains the top-right corner of R. Otherwise, we call the leader an *outward* leader. The leaders incident to ℓ and ℓ' in Figure 2(a), are inward and outward leaders, respectively.

Assume that $\mathcal{I}$ has an affirmative solution $\mathcal{S}$ and let C be a corresponding xy-separating curve. Let $up(C)$ be the polygonal region above C bounded by $R(t)$ and $R(l)$. Similarly, let $right(C)$ be the polygonal region to the right of C bounded by $R(b)$ and $R(r)$. By C_u (resp., C_r) we denote the xy-separating curve that minimizes the area of $up(C_u)$ (resp., $right(C_r)$),

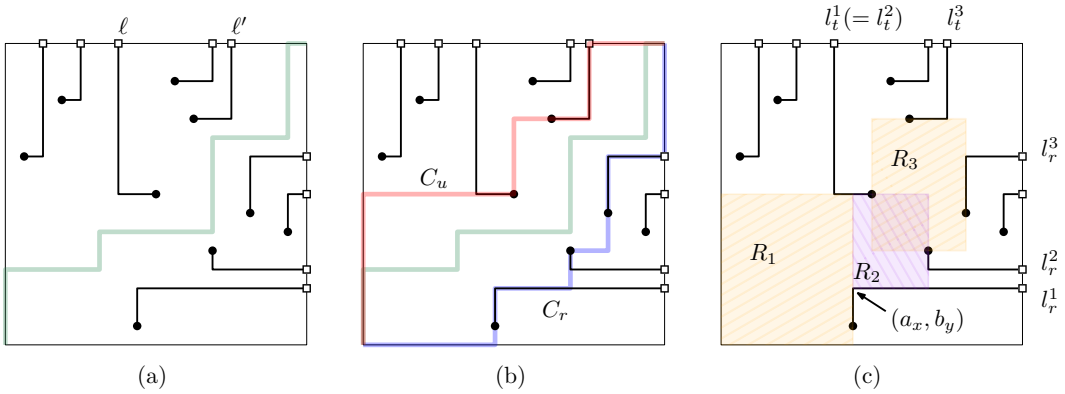


Figure 2 (a) An xy-separated solution to a 2-sided boundary labeling. The xy-separating curve C is shown in light-green. (b) Illustration for the curves C_u and C_r. (c) $\mathcal{R}$.

e.g., see Figure 2(b). For a point p, let p_x and p_y be its x and y-coordinates, respectively. Given C_u and C_r, we define a sequence of rectangles $\mathcal{R} = (R_1, R_2, \ldots, R_k)$ as follows:

- Each rectangle is a maximal rectangle between C_u and C_r.
- The bottom-left corner of R_1 coincides with that of R.
- For $i > 1$, we first consider R_{i-1}. Since R_{i-1} is maximal, the top and right sides of R_{i-1} must be determined by a pair of leaders, e.g., see R_1 in Figure 2(c). Denote these leaders by ℓ_t^{i-1} and ℓ_r^{i-1}, respectively. Let $a \in \ell_t^{i-1}$ be the rightmost point of ℓ_r^{i-1} on the top side of R_{i-1}, and let $b \in \ell_r^{i-1}$ be the topmost point on the right side of R_{i-1}. We define R_i to be the maximal empty rectangle with the bottom-left corner at (a_x, b_y) and the sides bounded by C_u and C_r.

2.2 Algorithm

The idea of the algorithm is to employ a dynamic programming algorithm based on the idea of finding the optimal rectangle sequence $\mathcal{R}$. Note that for any rectangle $R_j \in \mathcal{R}$, we can think of a subproblem $\lambda(R_j)$ that seeks a solution including the leaders ℓ_t^j and ℓ_r^j. More formally, $\lambda(R_j)$ is an instance of the 2-sided boundary labeling problem for which the rectangle $B(R_j)$ corresponding to this problem is determined by the vertical segment of ℓ_t^j, the horizontal segment of ℓ_r^j as well as the top and right sides of the rectangle R; see the gray rectangle in Figure 3(a). It is straightforward to add a dummy rectangle R_0 with corresponding leaders ℓ_t^0 and ℓ_r^0 such that $\lambda(R_0)$ represents the original 2-sided boundary labeling problem; e.g., see Figure 3(b).

Given R_j, we try to find R_{j+1} by checking all possible candidate rectangles. For convenience, we defer the details of finding all candidate rectangles, and focus on the computation of the solution cost (sum of leader length) assuming that we have found R_{j+1}. Figure 3(c) illustrates such a scenario. Let $R^t_{j,j+1}$ be the region bounded by the lines determined by the vertical segments of ℓ_t^j and ℓ_t^{j+1}, the horizontal segment of ℓ_t^j, and $R(t)$. Define $R^r_{j,j+1}$ symmetrically, e.g., see the top of Figure 4(i). Observe that $\lambda(R^t_{j,j+1})$ is a 1-sided boundary labeling problem with leaders ℓ_t^j and ℓ_t^{j+1}. In other words, since R_{j+1} is an empty rectangle, all the ports between ℓ_t^j and ℓ_t^{j+1} must be connected to some site interior to $R^t_{j,j+1}$. We define $\lambda(R^r_{j,j+1})$ symmetrically. It is now straightforward to express the solution of $\lambda(R_j)$ in terms of the solutions of $\lambda(R^t_{j,j+1}), \lambda(R^r_{j,j+1})$, and $\lambda(R_{j+1})$.

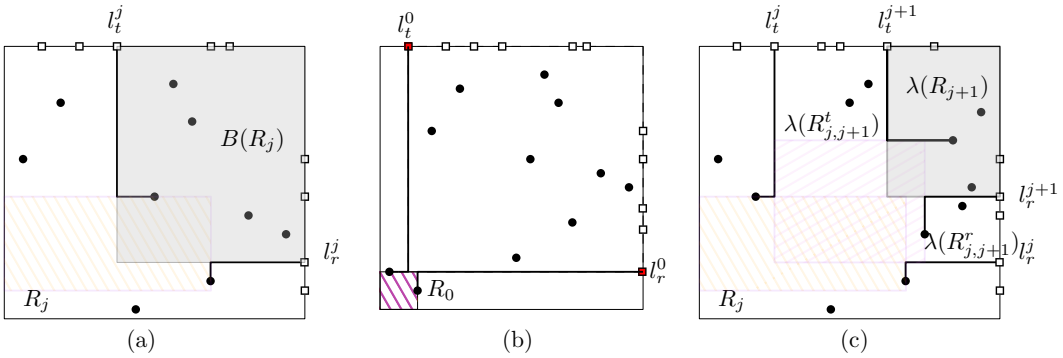


Figure 3 Illustration for the dynamic programming algorithm.

For any leader l, we denote its length by $|l|$. Let $|\lambda(R_j)|$ be the sum of the leader lengths in an optimal solution of $\lambda(R_j)$ (excluding the lengths of ℓ_t^j and ℓ_r^j). Let $ports(B(R_j))$ and $sites(B(R_j))$ be the number of ports and sites interior to $B(R_j)$, excluding those that are incident to ℓ_t^j and ℓ_r^j. We now have the following recursive formula, where $\mathcal{C}$ denotes the set of candidate rectangles.

$$|\lambda(R_j)| = \begin{cases} \infty, & \text{if } ports(B(R_j)) \neq sites(B(R_j)). \\ (|\ell_t^j| + |\ell_r^j|) + & \\ \min\limits_{R_{j+1} \in \mathcal{C}} \{|\lambda(R^t_{j,j+1})| + |\lambda(R^r_{j,j+1})| + |\lambda(R_{j+1})|\}, & \text{otherwise.} \end{cases}$$

Finding candidate rectangles. Given a rectangle R_j, we now describe how to find a set of candidate rectangles that must include R_{j+1}. Recall that we can compute the bottom-left corner (a_x, b_y) of R_{j+1} from R_j. Figures 4(a)–(d) illustrate the scenarios where ℓ_t^j and ℓ_r^j are inward. The point (a_x, b_y) is marked with a cross. We claim that the top side or the right side of R_{j+1} must contain a site (Lemma 3). We will use the following result of Benkert et al. [7] to prove Lemma 3.

▶ **Lemma 2** (Benkert et al. [7]). *For any solution S to a 1-bend 1-sided boundary labeling problem that minimizes the total leader length (possibly with crossings), there exists a crossing-free labeling with the total leader length at most the total leader length of S.*

▶ **Lemma 3.** *The top side or the right side of R_{j+1} must contain a site.*

Proof. Suppose for a contradiction that neither the top nor the right side of R_{j+1} contains a site. We now consider four cases.

Case 1 (both ℓ_t^{j+1} and ℓ_r^{j+1} are inward): In this case the leaders ℓ_t^{j+1} and ℓ_r^{j+1} must intersect (see Figure 4(e)), which contradicts that the underlying solution is crossing-free.

Case 2 (ℓ_t^{j+1} is inward and ℓ_r^{j+1} is outward): If ℓ_r^j is outward, then it must intersect ℓ_r^{j+1} (see Figure 4(f)). Therefore, the leader ℓ_r^j must be inward, as illustrated in Figure 4(g). Note that by our general position assumption, the 'y-intervals' determined by the vertical segments of ℓ_r^j and ℓ_r^{j+1} must overlap. Consequently, by swapping the site assignments, we can obtain a solution (possibly with crossings) with strictly smaller total leader length. Figure 4(i) illustrates such a scenario. By Lemma 2, we can replace this labeling of $\lambda(R_{j,j+1})$ with a crossing free labeling that lies inside $R^r_{j,j+1}$ and does not increase the total leader length, e.g., see Figure 4(j). Note that the total leader length of the resulting solution would be strictly smaller, contradicting that the current solution is optimal.

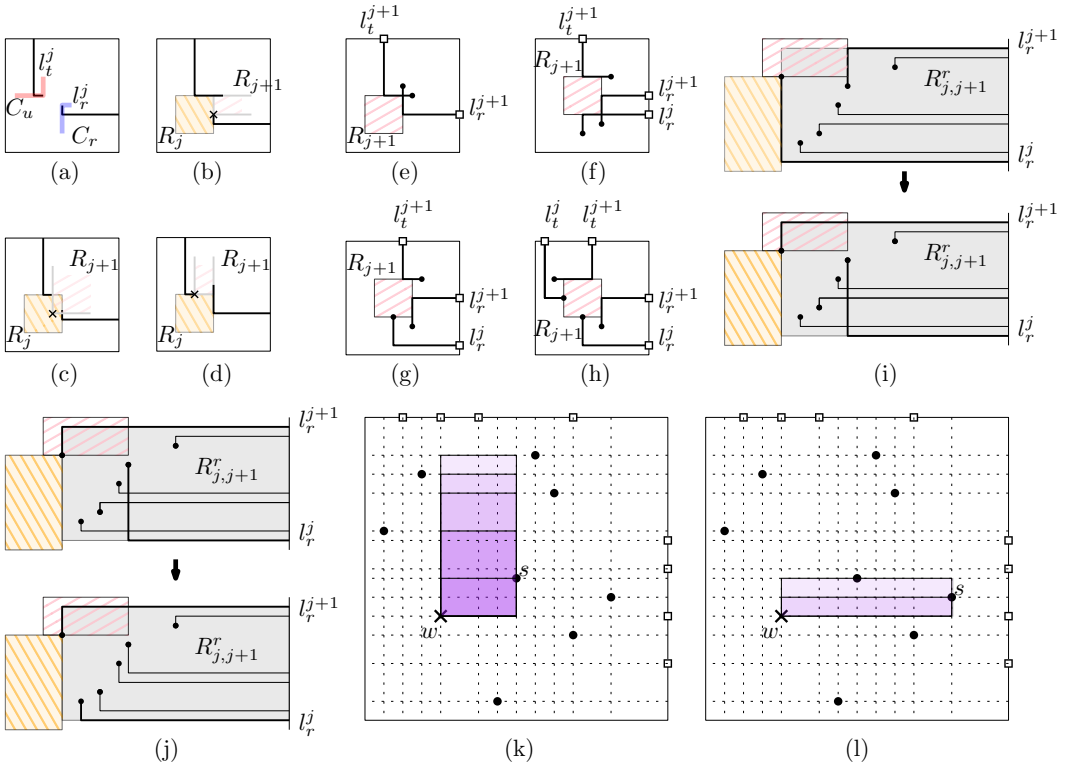


Figure 4 Illustration for (a)–(d) (a_x, b_y), (e)–(j) Lemma 3, and (k)–(l) candidate rectangles.

Case 3 (ℓ_t^{j+1} is outward and ℓ_r^{j+1} is inward): This case is symmetric to Case 2.
Case 4 (both ℓ_t^{j+1} and ℓ_r^{j+1} are outward): We can process this case in the same way as we did in Case 2. ◀

Recall that we know the bottom-left point w of R_{j+1}. We first assume that the right side of R_{j+1} contains a site. For every site s with $s_x > w_x$ and $s_y > w_y$, we consider all possible empty rectangles with bottom-left corner w, right side passing through s and the top side determined by a horizontal line passing through a site above s. Figures 4(k)–(l) illustrate the candidate rectangles for the bottom left point w. We then assume that the top side of R_{j+1} contains a site, and find the candidate rectangles symmetrically. We can now obtain an upper bound on the distinct candidate rectangles.

▶ **Lemma 4.** *The overall number of distinct candidate rectangles is $O(n^3)$.*

Proof. For a particular bottom-left corner w, it may initially appear that there are $O(n^2)$ possible candidate rectangles to explore. But we can prove an $O(n)$ upper bound, as follows.

Let D be the region dominated by w; i.e., for each point $q \in D$, the x and y-coordinates of q are at least as large as those of w. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of sites in D (ordered by increasing y-coordinates) such that no site in S is dominated by any other site in D (except possibly for w). We may assume without loss of generality (see Lemma 3) that the right side of R_{j+1} contains a site. Since the proper interior of the rectangle is empty, for each s_i, where $1 \leq i \leq k$, we only need to consider a set of heights $H(s_i)$ that lie between s_i and s_{i+1} (or, between s_i and $R(t)$ when $i = k$). For every pair of sites $\{s, s'\} \in S$, we have the property that neither s nor s' dominates the other. Therefore, we have $H(s) \cap H(s') = \emptyset$, $\sum_i H(s_i) = O(n)$, and thus a linear number of candidate rectangles for w.

The number of possible intersections (i.e., bottom-left corners) among the horizontal and vertical lines passing through the ports and sites is $O(n^2)$. Therefore, the number of distinct candidate rectangles that may appear over the run of the algorithm is $O(n^3)$. ◀

Data structures and time complexity. If we use an $O(n^2) \times O(n^2)$ dynamic programming table and compute each entry by checking $O(n)$ candidate rectangles, then we need at least $O(n^5)$ time. To improve the running time to $O(n^3 \log n)$, we preprocess the input. For every possible matching of a pair of ports (on the same side of R) to a pair of sites, we compute and store the solution to the corresponding 1-sided boundary labeling problem. Since there are $O(n^4)$ such 1-sided problems, and each of them can be answered in $O(n \log n)$ time [7], this takes $O(n^5 \log n)$ time. We first show how to reduce this preprocessing time to $O(n^3 \log n)$.

Consider a subproblem $\lambda(R^t_{j,j+1})$. Such a problem can easily be expressed by the ports and sites incident to ℓ^j_t and ℓ^{j+1}_t. Here we encode $\lambda(R^t_{j,j+1})$ in a slightly different way. We use the parameters p, p', α, β, where p, p' are the ports incident to ℓ^j_t and ℓ^{j+1}_t, α is either ∞ or the y-coordinate of a site that indicates the top side of an "empty rectangle", which is used in our preprocessing, and β is the 'type' of $\lambda(R^t_{j,j+1})$. We will express $\lambda(R^t_{j,j+1})$ as $S(p, p', \alpha, \beta)$. In the following we describe the details of $S(p, p', \alpha, \beta)$.

Note that to solve $\lambda(R^t_{j,j+1})$ affirmatively, we need exactly as many free sites as the number of ports between p and p'. Thus for any subproblem, if the number of free sites and free ports interior to $R^t_{j,j+1}$ do not match, then we can immediately return a negative answer. We assume that the points and ports are stored in an orthogonal range counting data structure (with $O(n \log n)$-time preprocessing) such that given an orthogonal rectangle, one can report the number of ports and points interior to the rectangle in $O(\log n)$ time [9]. We only focus on those instances that have the same number of free sites and ports, and express them in the form $S(p, p', \alpha, \beta)$.

Let s, s' be the sites that are incident to ℓ^j_t and ℓ^{j+1}_t, respectively. By the property of the optimal solution, we may assume that $s_y < s'_y$. We define $\lambda(R^t_{j,j+1})$ as having Type 1, 2, 3 or 4 depending on whether s, s' belongs to $R^t_{j,j+1}$ or not.

Type 1 (both s, s' are outside $R^t_{j,j+1}$): In this case the rectangle determined by the bend points of ℓ^j_t and ℓ^{j+1}_t must be empty (i.e., the gray region in Figure 5(a)). We set α to be ∞, and β to be 1. During the algorithm execution, if $\lambda(R^t_{j,j+1})$ is of Type 1, then we will seek a solution to $S(p, p', \infty, 1)$.

Note that for any instance of the form $S(p, p', \infty, 1)$, we can determine in $O(1)$ time[2] the point s'' such that the rectangle B determined by p, p', s'' contains an equal number of free ports and sites. Note that the solution to the labeling problem inside B will be equivalent to that of $\lambda(R^t_{j,j+1})$. We will precompute the solutions of $S(p, p', \infty, 1)$ so that $\lambda(R^t_{j,j+1})$ can be answered in $O(1)$ time by a table look-up. This general idea of answering a problem $\lambda(\cdot)$ using $S(\cdot)$ applies also to the other types; i.e., Types 2, 3 and 4.

Type 2 (both s, s' are inside $R^t_{j,j+1}$): In this case the rectangle determined by the bend point of ℓ^{j+1}_t and s must be empty; see Figures 5(c). (Notice that the case shown in Figure 5(b) is not possible in an optimal solution because re-routing p' to s and p to s' would result in a feasible solution with a smaller total leader length.) We thus set α to be $y(s')$, and β to be 2. Observe now that given $S(p, p', \alpha, 2)$, we can find both s and s' in $O(1)$ time[2] by counting the number of ports between p and p', and using α.

[2] It is straightforward to preprocess the ports and sites in $O(n^3)$ time in a data structure to answer such queries in $O(1)$ time.

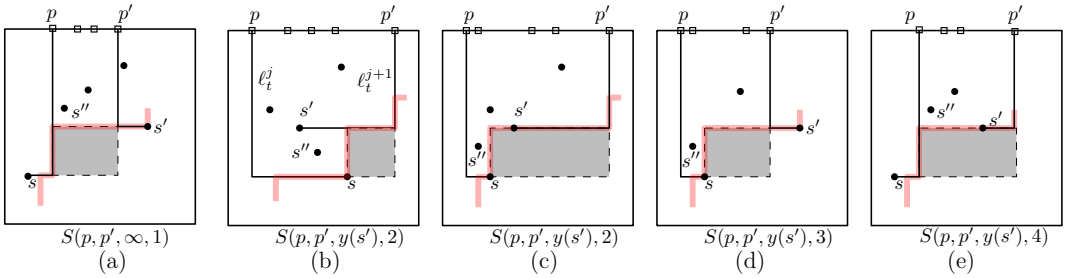


Figure 5 (a)–(e) Illustration for different Types of subproblems.

Type 3 ($s \in R^t_{j,j+1}$ and $s' \notin R^t_{j,j+1}$): In this case the rectangle determined by the bend point of ℓ_t^{j+1} and s must be empty (Figure 5(d)). We thus set α to be $y(s')$, and β to be 3. Given $S(p, p', \alpha, 3)$, we can recover s and s' using the range counting data structures[2]. The same argument holds even when $s_x > p'_x$.

Type 4 ($s \notin R^t_{j,j+1}$ and $s' \in R^t_{j,j+1}$): In this case the rectangle determined by the bend points of ℓ_t^j and ℓ_t^{j+1} must be empty (Figure 5(e)). We thus set α to be $y(s')$, and β to be 4. Given $S(p, p', \alpha, 4)$, we can recover s' using α. Here we do not need to find s since the solution must lie inside the rectangle determined by p, p' and s'.

▶ **Lemma 5.** *The solution to the problems $S(p, p', \alpha, \beta)$ can be computed in $O(n^3 \log n)$ time.*

Proof. Since there are $O(n)$ possible choices for each of p, p', α, and a constant number of choices for β, we have at most $O(n^3)$ subproblems. We can employ a dynamic programming to compute the solution to these problems. The idea is to select the bottommost free point s'' and connect it to a port p'' between p and p'. This splits the problem into two subproblems, which can again be expressed in the form $S(p, p', \alpha, \beta)$. Such a split may generate a new type of subproblem $\mathcal{Q}$, where ℓ_t^j has a shorter height than that of ℓ_t^{j+1}. Since ℓ_t^j was initially incident to s'', we can process $\mathcal{Q}$ as follows: For every pair of ports, we use Benkert et al.'s [7] algorithm to precompute the solution to the boundary labeling problem inside the stripe bounded by the vertical lines through p, p'. If there are k ports between p and p', then we use the topmost k sites in the stripe (if it exists). This preprocessing takes $O(n^3 \log n)$ time. To answer $\mathcal{Q}$, we use the precomputed solution for the corresponding stripe.

Since the number of choices for p'' is at most n, we can compute an entry of the dynamic programming table by a linear number of table look-up. Since the number of entries is $O(n^3)$, the running time is bounded by $O(n^4)$. An involved analysis shows that there is only $O(1)$ candidate choices for p'', and these candidates can be found in $O(\log n)$ time. The full version of the paper [8] includes the details. Since an entry of the dynamic programming table can now be computed using $O(1)$ number of table look-ups, the running time reduces to $O(n^3 \log n)$. ◀

▶ **Theorem 6.** *Given a 1-bend 2-sided boundary labeling problem with $O(n)$ sites and labels, one can find a labeling (if exists) that minimizes the total leader length in $O(n^3 \log n)$-time.*

Proof. Every subproblem $\lambda(R_j)$ can be defined by a pair of leaders, and hence we can define an $O(n^2) \times O(n^2)$ table T to store the solutions to the subproblems. To compute an entry of the table T, we look at a set of candidate rectangles with two nice properties. First, all these rectangles have the same bottom-left corner, and second, none of these rectangles can be a candidate rectangle for any other entry of T. Therefore, the number of 'candidate

rectangle queries' to fill all the entries of T is bounded asymptotically by the number of distinct candidate rectangles, which is $O(n^3)$ (by Lemma 4). Since we do not recompute solutions, and the table look-up takes $O(1)$ time, the total running time is bounded by $O(n^4)$, which dominates the preprocessing time.

Observe that the complexity $O(n^4)$ comes from considering all possible pairs of leaders, whereas only $O(n^3)$ options are relevant (by Lemma 4). Therefore, instead of a table, we can keep the relevant entries in a dynamic binary search tree, which increases the cost for solution look-up to $O(\log n)$, but limits the time for both the memory initialization and look-up queries to $O(n^3 \log n)$. Thus the total running time improves to $O(n^3 \log n)$. ◀

3 Relating Boundary Labeling to Outerstring Graphs

In this section, we reduce the boundary labeling problem to the independent set problem on a class of weighted geometric intersection graphs in the plane called outerstring graphs. We show that if one can discretize a boundary labeling problem such that the number of candidate leaders is a polynomial in n, then our approach will yield a polynomial-time algorithm for the problem.

An *outerstring graph* is an intersection graph of a set of curves in the Euclidean plane that lie inside a polygon such that one of the endpoints of each curve is attached to the boundary of the polygon. Keil et al. [16] gave an $O(N^3)$-time algorithm for the maximum-weighted independent set problem on outerstring graphs. The algorithm requires an outerstring graph as an input, where each curve is given as a polygonal line (i.e., a chain of straight line segments) and N is the number of segments in the representation. We show that by discretizing the boundary labeling problem and assigning an appropriate weight to each candidate leader, one can reduce the boundary labeling problem to the maximum-weighted independent set problem on outerstring graphs. Here, as an example, we show the reduction for the boundary labeling problem using *po*- and *opo*-leaders in the presence of obstacles.

Boundary labeling with orthogonal obstacles. Fink and Suri [11] gave $O(n^9)$ and $O(n^{21})$-time algorithms for the opposite 2-sided boundary labeling with *po*- and *opo*-leaders, respectively. Our approach will yield $O(n^6)$ and $O(n^{12})$-time algorithms for *po*- and *opo*-leaders, respectively, irrespective of the labeling model (opposite, adjacent, or for any port distribution on the boundary). For the opposite 2-sided case, the running time reduces to $O(n^6)$ and $O(n^9)$ (for *po*- and *opo*-leaders, respectively). This will settle the time complexity question of 1-bend 3- and 4-sided boundary labeling [17]. In the rest of this section, we relax the general position assumption and denote n to be the total number of sites and obstacle vertices.

First consider the case of *po*-leaders. Let I be an instance of the boundary labeling problem. Given a site and a port, there is at most one way of connecting them. Let M denote the set of all possible leaders that do not intersect any obstacle. Then $|M| \in O(n^2)$. It is straightforward to compute M in $O(n^3)$ time. Observe that each leader $l \in M$ can be viewed as an outerstring, and let $st(l)$ be the corresponding outerstring. Let $|l|$ be the length of the leader l, and define $x := \max_{l \in M} |l|$ and $y := \min_{l \in M} |l|$. Let C be a number such that $C > nx - (n-1)y > 0$. For each leader $l \in M$, we assign a weight $w(st(l))$ to $st(l)$, where $w(st(l)) := C - |l|$. The following lemma and Keil et al.'s [16] algorithm lead us to the results for *po*-leaders (Theorem 8).

▶ **Lemma 7.** *I has a feasible solution with total leader length L if and only if the corresponding outerstring graph G_I has a feasible solution with total weight $(nC - L)$.*

Proof. A feasible solution S of I with total leader length L gives a feasible solution for G_I with total weight

$$\sum_{l \in S} w(st(l)) = \sum_{l \in S} (C - |l|) = nC - \sum_{l \in S} |l| = (nC - L).$$

We now assume that G_I has a feasible solution S' with total weight $W = (nC - L)$, and show that the corresponding leaders S yields a feasible solution of I of total leader length L. Since S' is an independent set, the leaders in S are crossing-free, as well as no site or port is incident to more than one leader. It now suffices to show that every site is connected to a string, i.e., $|S| = n$ and the total leader length is L. Observe that $W = nC - L \geq nC - nx > nC - (n-1)y - C = (n-1)(C-y)$. If $|S| < n$, then the total leader length is at most $(n-1)x$, and S' has weight at most $(n-1)(C-y)$, which contradicts that $W > (n-1)(C-y)$. Therefore, $|S| = n$, and we have

$$W = \sum_{s \in S'} w(s) = \sum_{s \in S'} (C - |l_i|) = nC - \sum_{l \in S'} |l|.$$

Since $W = (nC - L)$, we have $\sum_{l \in S'} = L$. ◀

▶ **Theorem 8.** *The boundary labeling problem can be solved in $O(n^6)$ time using po-leaders, for both adjacent and opposite sided models, even in the presence of obstacles (where n is the total number of sites and vertices of the obstacles).*

Consider now the case for *opo*-leaders. For opposite 2-sided case, Fink and Suri [11] showed that one can discretize the problem such that if there exists a feasible solution, then there is one where the x-coordinate of the middle segment of every leader lies in the set of all x-coordinates of the sites and obstacle vertices. Therefore, we have $O(n)$ potential leaders for each port-site pair, and thus $O(n^3)$ leaders in total. Hence applying Keil et al.'s [16] algorithm gives a running time of $O(n^9)$.

The discretization of [11] does not apply to the 3- and 4-sided case. However, consider a grid H determined by the axis-aligned lines through the ports, sites and obstacle vertices. For each pair of consecutive parallel lines of H, place a set of n parallel lines in between. Let the resulting grid be H'. If there is a feasible solution to the boundary labeling problem, then for any pair of consecutive parallel vertical lines ℓ, ℓ' (similarly for horizontal) of H, we can have at most n middle vertical segments of the leaders. We thus can distribute them by moving horizontally to the n lines of H' (e.g., see [11]), which does not change the total leader length. By construction, there is no site, port or obstacle vertex between ℓ and ℓ'. Hence such a modification can be performed without introducing any crossing. Since H' is an $O(n^2) \times O(n^2)$ grid and since we have $O(n^2)$ potential leaders for each port-site pair, the number of candidate leaders is $O(n^4)$. Hence applying Keil et al.'s [16] algorithm gives a running time of $O(n^{12})$.

▶ **Theorem 9.** *The adjacent boundary labeling problem can be solved in $O(n^{12})$ time using opo-leaders, even in the presence of obstacles (where n is the total number of sites and vertices of the obstacles). For opposite 2-sided models, the running time reduces to $O(n^9)$.*

Sliding ports and bend minimization. The outerstring-graph approach can also be applied to the sliding port model, where each label is assigned a distinct interval on the boundary of R and a site can be connected to any point of an interval. The goal here is to minimize the total leader length or the number of bends. We only need to discretize the problem such

that the number of strings that we need to consider is a polynomial in n. Define H to be a grid determined by the axis-aligned lines through sites, interval boundaries and obstacle vertices. Construct H' from H by introducing for every pair of consecutive parallel lines of H, a set of $2n$ parallel lines in between.

The grid H' can be used to discretize the problem, as follows. The segments incident to the sites are already on H. Consider now a vertical (similarly for horizontal) segment ℓ that is incident to an interval I, but not incident to any site. Let ℓ' and ℓ' be a pair of consecutive horizontal lines of H such that ℓ lies between them. There can be at most $2n$ horizontal lines between ℓ, ℓ', which we can distribute to the lines of H' by moving vertically (e.g., see [11]). Since there cannot be any site, interval boundary or obstacle vertex between ℓ, ℓ', such a modification neither introduce crossings nor increase the number of total bends. By the construction of H, the boundary of R between ℓ, ℓ' lies in the interval I. Hence ℓ will still be incident to I. Finally, the middle segments of the leaders can be processed in the same way as we did for Theorem 9. It is straightforward to observe that the number of potential strings is a polynomial in n. We can now assign certain weights to these strings such that the maximum-weight independent set of the corresponding outerstring graph yields a minimum-bend solution for the boundary labeling problem.

We first consider the case of *po*-leaders. Let I be an instance of this problem. Consider the set M of outerstrings as before. For each outerstring $st(l) \in M$, we assign the weight $w(st(l))$, where

$$w(st(l)) = \begin{cases} n+2, & \text{if } l \text{ has no bends.} \\ n+1, & \text{if } l \text{ has one bend.} \end{cases} \tag{1}$$

This forms our instance G_I of an outerstring graph on which we solve the maximum-weighted independent set problem by running Keil et al.'s algorithm [16].

▶ **Lemma 10.** *Let I be an instance of the boundary labeling problem with po-leaders. Then I has a feasible solution with k bends if and only if the instance G_I has a feasible solution W with total weight at least $(n^2 + 2n - k)$.*

Proof. Let S be a feasible solution of I. Clearly, the strings corresponding to the leader of S' is a feasible solution for G_I. Let k be the total number of bends in S. Then the weight of S' is $\sum_{l \in S} w(l) \geq (n+1)k + (n+2)(n-k) = n^2 + 2n - k$.

Assume now that G_I has a feasible solution S with weight at least $n^2 + 2n - k$. Let S' be the corresponding set of leaders in I. Since S is an independent set, a port or site can be incident to at most one leader of S. If a site is not connected to any port in S', then at most $(n-1)$ sites are incident to a leader. Since the maximum weight of a leader can be at most $(n+2)$, the weight of S is at most $(n-1)(n+2) = (n^2+n-2)$, which is a contradiction since the weight of S is at least $n^2 + 2n - k > (n^2 + n - 2)$ (because $n \geq k$). Therefore, $|S'| = n$.

It now remains to show that the weight of S' is at most k. Suppose for a contradiction that S' has at least $(k+1)$ *po*-leaders. Therefore, the weight of S is at most $(n+1)(k+1) + (n+2)(n-k-1) = n^2 + 2n - (2k+1) < (n^2 + 2n - k)$, which is a contradiction that the weight of S is at least $(n^2 + 2n - k)$. ◀

Now, we consider the case of *opo*-leaders. Let I be an instance of this problem. Consider the set M of outerstrings as before. For each outerstring $st(l) \in M$, we assign the weight

$w(st(l))$ as follows:

$$w(st(l)) = \begin{cases} \alpha + 3, & \text{if } l \text{ has no bends,} \\ \alpha + 2, & \text{if } l \text{ has one bend,} \\ \alpha + 1, & \text{if } l \text{ has two bends.} \end{cases} \tag{2}$$

Here, $\alpha = 2n$.

▶ **Lemma 11.** *Let I be an instance of the boundary labeling problem with opo-leaders. Then I has a feasible solution with k bends if and only if the instance G_I has a feasible solution W with total weight at least $(\alpha n + 3n - k)$.*

Proof. Let S be a feasible solution of I. Clearly, the strings corresponding to the leader of S' is a feasible solution for G_I. Let k be the total number of bends in S, and let k_1 and k_2 be the number of strings with 1-bend and 2-bends, respectively. Therefore, $k_1 + 2k_2 = k$, and the weight of S' is $\sum_{l \in S} w(l) = (\alpha+2)k_1 + (\alpha+1)k_2 + (\alpha+3)(n - k_1 - k_2) = \alpha n + 3n - k_1 - 2k_2 = \alpha n + 3n - k$.

Assume now that G_I has a feasible solution S with weight at least $(\alpha n + 3n - k)$. Let S' be the corresponding set of leaders in I. Since S is an independent set, a port or site can be incident to at most one leader of S. If a site is not connected to any port in S', then at most $(n-1)$ sites are incident to a leader. Since the maximum weight of a leader can be at most $(\alpha + 3)$, the weight of S is at most $(n-1)(\alpha+3) = (\alpha n + 3n - \alpha - 3)$, which is a contradiction since the weight of S is at least $\alpha n + 3n - k > (\alpha n + 3n - \alpha - 3)$ (because $\alpha = 2n \geq k$). Therefore, $|S'| = n$.

It now remains to show that the leaders of S' has at most k bends. Suppose for a contradiction that S' has at least k_1 *po*-leaders and k_2 *opo*-leaders such that $k_1 + 2k_2 \geq k+1$. Therefore, the weight of S is at most $(\alpha + 2)k_1 + (\alpha + 1)k_2 + (\alpha + 3)(n - k_1 - k_2) = \alpha n + 3n - (k_1 + 2k_2) - (2k_1 + k_2) + (2k_1 + k_2) = \alpha n + 3n - (k_1 + 2k_2)$. Since $k_1 + 2k_2 \geq k + 1$, the weight of S is strictly less than $\alpha n + 3n - k$, which is a contradiction. ◀

By Lemmas 10 and 11, we have the following theorem (which settles two open questions of [20, Table 23.1]).

▶ **Theorem 12.** *A boundary labeling that minimizes the total number of bends can be computed (if exists) in polynomial time for both adjacent and opposite models (with sliding ports, po and opo-leaders), even in the presence of obstacles.*

4 Conclusion

The most natural directions for future research is to improve the time complexity of our algorithm for the 1-bend adjacent 2-sided model. A number of intriguing questions follow: Can we find a non-trivial lower bound on the time-complexity? Is the problem 3-sum hard or, as hard as 'sorting $X + Y$'? Can we check the feasibility in near-linear time? It would also be interesting to find fast approximation algorithms for boundary labeling problems.

References

1 Alexander Wolff and Tycho Strijk. The map-labeling bibliography. `http://i11www.ira.uka.de/map-labeling/bibliography/`. Online; accessed 10 February, 2018.

2 Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. On the readability of boundary labeling. In *23rd International Symposium Graph Drawing and Network Visualization (GD 2015), Los Angeles, CA, USA*, pages 515–527, 2015.

3 Lukas Barth, Benjamin Niedermann, Martin Nöllenburg, and Darren Strash. Temporal map labeling: A new unified framework with experiments. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, pages 23:1–23:10. ACM, 2016.

4 Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. Many-to-one boundary labeling with backbones. *J. Graph Algorithms Appl.*, 19(3):779–816, 2015.

5 Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010.

6 Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comput. Geom.*, 36(3):215–236, 2007.

7 Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009.

8 Prosenjit Bose, Paz Carmi, J. Mark Keil, Saeed Mehrabi, and Debajyoti Mondal. Boundary labeling for rectangular diagrams. *CoRR*, abs/1803.10812, 2018. URL: `https://arxiv.org/abs/1803.10812`.

9 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin Heidelberg, 2008.

10 Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M. E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 148–157, 1997.

11 Martin Fink and Subhash Suri. Boundary labeling with obstacles. In *Proceedings of the 28th Canadian Conference on Computational Geometry (CCCG)*, pages 86–92, 2016.

12 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the Seventh Annual Symposium on Computational Geometry (SoCG)*, pages 281–288. ACM, 1991.

13 Herbert Freeman. An expert system for the automatic placement of names on a geographic map. *Inf. Sci.*, 45(3):367–378, 1988.

14 GLEAM. `http://www.greatlakesmapping.org/`. Online; accessed 10 February, 2018.

15 Stephen A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

16 J. Mark Keil, Joseph S. B. Mitchell, Dinabandhu Pradhan, and Martin Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Comput. Geom.*, 60:19–25, 2017.

17 Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. Multi-sided boundary labeling. *Algorithmica*, 76(1):225–258, 2016.

18 Benjamin Niedermann, Martin Nöllenburg, and Ignaz Rutter. Radial contour labeling with straight leaders. In *2017 IEEE Pacific Visualization Symposium (PacificVis 2017), Seoul, South Korea*, pages 295–304, 2017.

19 Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS)*, pages 310–319, 2010.

20 Alexander Wolff. Graph drawing and cartography. In Roberto Tamassia, editor, *Handbook of graph drawing and visualization*, chapter 23, pages 697–736. CRC Press, 2014.

21 Steven Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.

22 Steven Zoraster. Practical results using simulated annealing for point feature label placement. *Cartography and GIS*, 24(4):228–238, 1997.

Gathering by Repulsion

Prosenjit Bose[1]
School of Computer Science, Carleton University, Canada
jit@scs.carleton.ca

Thomas C. Shermer
School of Computing Science, Simon Fraser University, Canada
shermer@sfu.ca

Abstract

We consider a repulsion actuator located in an n-sided convex environment full of point particles. When the actuator is activated, all the particles move away from the actuator. We study the problem of gathering all the particles to a point. We give an $O(n^2)$ time algorithm to compute all the actuator locations that gather the particles to one point with one activation, and an $O(n)$ time algorithm to find a single such actuator location if one exists. We then provide an $O(n)$ time algorithm to place the optimal number of actuators whose sequential activation results in the gathering of the particles when such a placement exists.

2012 ACM Subject Classification Mathematics of computing → Graph theory, Theory of computation → Design and analysis of algorithms, Theory of computation → Computational geometry

Keywords and phrases polygon, kernel, beacon attraction

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.13

1 Introduction

In this paper, we consider some basic questions about *movement by repulsion*. Here a point actuator repels particles, or put another way, particles move so as to locally maximize their distance from the actuator. This problem models magnetic repulsion, movement of floating objects due to waves, robot movement (if robots are programmed to move away from certain stimuli), and crowd movement in an emergency or panic situation. It is, in one sense, the opposite of movement by attraction, which has recently been an active topic of research [2, 3, 11, 10, 14, 9, 1, 8].

1.1 Related work

We initiate the study of repulsion in polygonal settings. The closest comparable work is the work on attraction. Although attraction and repulsion have a similar definition, each has a distinct character. Attraction as it has been studied is mainly a two-point relation: a point p attracts a point q if q, moving locally to minimize distance to p, eventually reaches p. In replusion, p cannot repulse q to itself; p must always repulse q to some other point r. Thus repulsion is a three-point relation.

In attraction, if a particle is attracted onto an edge by a beacon, it is pulled towards the point p where there is a perpendicular from the beacon to the line through the edge. If p is on the edge, this creates a stable minimum at p, and particles accumulate at such mimima. As well, particles can accumulate on some convex vertices.

[1] Research supported in part by NSERC.

In repulsion, if a particle is repelled onto an edge by a repulsion actuator, it is pushed away from the point p with the perpendicular to the actuator. This implies that p is an *unstable* maximum. We forbid particles from stopping at unstable maxima, so in repulsion the only accumulation points will be convex vertices. We elaborate further on our model in Subsection 1.2.

In this article, we highlight some of the similarities as well as distinctions between these two concepts. For instance, Biro [2] designed an $O(n^2)$ time algorithm for computing the *attraction kernel* of a simple n-vertex polygon P; these are all points $p \in P$ that attract all points $q \in P$. The closest counterpart of this for repulsion which we call the *repulsion kernel* of a polygon P, which is all points $p \in P$ such that there exists a point $r \in P$ such that p repels all points in P to r. We give an $O(n^2)$ time algorithm to compute the repulsion kernel of an n-vertex convex polygon, and an $O(n)$ time algorithm to find a single-point in the repulsion kernel or report that the kernel is empty.

Both the attraction kernel and the repulsion kernel are concerned with the problem of gathering particles to a point. When the repulsion kernel is empty, it may be the case that we can still gather all particles to a point using more than one repulsion actuator. In this vein, we prove that this is impossible in a polygon with three acute angles. In a convex polygon with at most two acute angles, two repulsion actuators are always sufficient and sometimes necessary. We then provide an $O(n)$ time algorithm to place the optimal number of actuators.

1.2 The model

We start with an n-vertex convex polygon P, which includes its interior. Before the activation of any repulsion actuator, there is a particle on every point of the polygon, including the boundary. During and after activation, we allow many particles to be on the same point; once two particles reach the same point, they travel identically, so we consider them to be one particle.

We restrict the location of the repulsion actuator to points in P; allowing the actuator to reside outside P leads to a variation of the problem in which convex polygons are easily dispensed.

See Figure 1 for an illustration of the following definitions. The activation of an actuator will cause all particles to move to locally maximize their distance from the actuator. This means that if a particle is in the interior of P, then it moves in a straight line away from the actuator's location. If a particle is on an edge of the polygon, then it proceeds along the edge in the direction that will further its distance from the active actuator. Once moving, a particle moves until it is stable and can no longer locally increase its distance from the actuator. Stable maxima happen at vertices where neither of the two edges allows movement away from the actuator. We call such vertices the *accumulation points* of the activation.

Unstable maxima happen when a particle is on an edge where one or both directions give no differential change of distance from the actuator; this happens only at the perpendicular projection of the actuator onto the edge (see Figure 1c). A particle at an unstable maximum will move off of it in a direction of no improvement and then will be able to increase the distance from the actuator by continuing in that direction. To maintain a deterministic model, we will assume that particles move counterclockwise around the polygon at unstable maxima if there is a choice of two directions of no improvement. However, the choice of counterclockwise motion is arbitrary, and does not affect our results.

We may activate actuators sequentially from several places inside the polygon. We would like for every activation of an actuator to be from a location without particles, but the particle-on-every-point model forbids this on the first activation. So, when we choose a

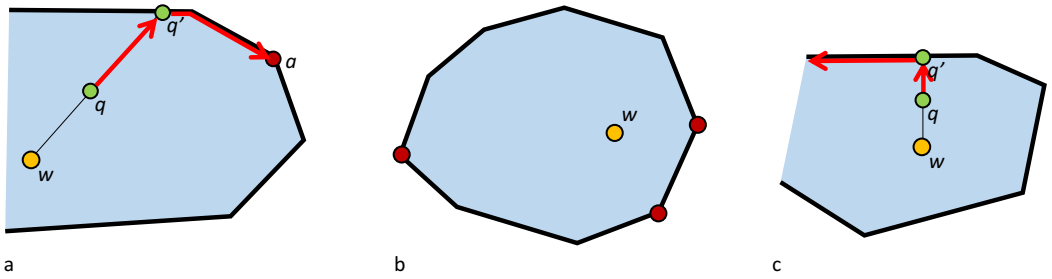


Figure 1 (a) An activation at w drives the particle at q away from w. On reaching an edge at q', it will continue to move away from w, until it reaches a local maximum of distance from w at a. (b) Accumulation points of an active actuator at w. (c) At an unstable maximum, such as q', particles will turn left.

location for the first actuator, we remove the particle at that location from the problem. For subsequent activations, however, we do require that the actuator's position be chosen from the points of the polygon without particles.

The main question we consider is when can we place a sequence of points such that repulsion from those points gathers all other points in the polygon to one point? When the replusion kernel is non-empty, one point is sufficient. In general, our goal is to minimize the number of sequential activations performed to gather all the particles to one point. If all the particles in a polygon can be gathered to a point with k sequential activations of actuators, we call the polygon *k-gatherable*. If this is not possible for any k, then we call the polygon *ungatherable*.

2 Background, notation, and terminology

2.1 General notation

We will use the convention that the vertices of P are $v_0, v_1, \ldots, v_{n-1}$ in counterclockwise order around the polygon. Vertex indices are taken modulo n, so $v_{-1} = v_{n-1}, v_0 = v_n, v_1 = v_{n+1}$, etc. Edges are denoted $e_0, e_1, \ldots e_{n-1}$, with e_i being the edge between v_i and v_{i+1}. The boundary of the polygon P will be denoted ∂P, and by $\partial P(p, q)$ we mean the part of ∂P from p counterclockwise to q. In reference to curves, line segments, or intervals, we use the usual parentheses to denote relatively open ends and square brackets to denote relatively closed ends. Thus $\partial P[p, q)$ is the boundary from p to q, including p but not q. Given three distinct points a, b, c in the plane, by $\angle abc$ we mean the counterclockwise angle between the ray from b to a and the ray from b to c.

2.2 Slabs and the three regions of an edge

Consider a polygon edge with particles covering it. When an actuator is activated, depending on its location relative to the edge, there are three possible effects on the particles: it drives them counterclockwise over the entire edge, it drives them clockwise over the entire edge, or it drives some of them clockwise and some of them counterclockwise (see Figure 2). In the latter case, a perpendicular from the edge to the actuator exists, and the particles clockwise of the perpendicular are driven clockwise, and the particles counterclockwise of the perpendicular are driven counterclockwise. The point where the perpendicular hits the edge is called a *split point*. We allow split points at the endpoint of an edge if a perpendicular from the endpoint to the actuator exists.

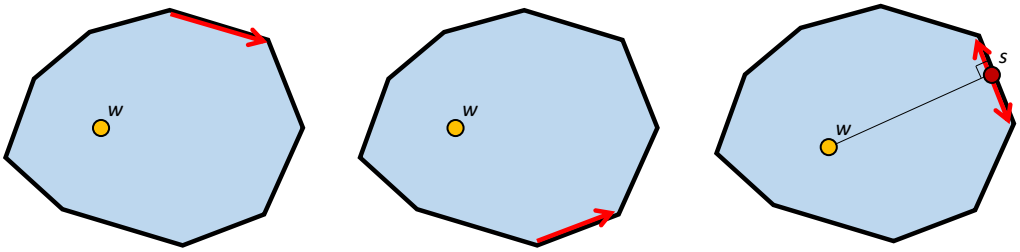


Figure 2 We use arrows in the diagrams to show the direction that the particles are driven. (a) The activation drives the particles (on the indicated edge) clockwise. (b) The activation drives the particles counterclockwise. (c) The activation splits the particles at s, driving some clockwise and some counterclockwise.

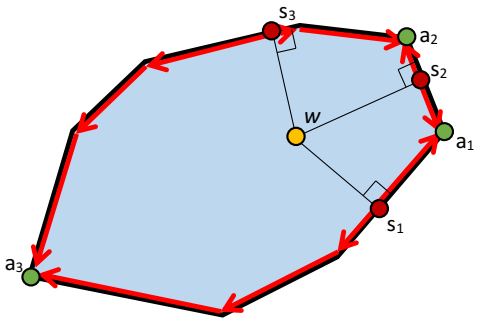


Figure 3 A flow diagram, showing the direction of particle movement, along with the accumulation points and split points, given an actuator at w.

We divide the inner halfplane of an edge e into three regions depending on what effect an activation in the region has on the particles on the edge. This is done by drawing interior-facing perpendiculars to the edge at each of its vertices. The regions are $R_{cw}(e)$, where an activation drives the particles clockwise, $R_{ccw}(e)$, where an activation drives the particles counterclockwise, and $S(e)$, where an activation drives some particles clockwise and some counterclockwise. We refer to $S(e)$ as the *slab* of e. The slab is closed on its boundaries, and $R_{cw}(e)$ and $R_{ccw}(e)$ are open where they meet $S(e)$.

2.3 Flow diagrams

Given a polygon P and a location w of an actuator, we may find the accumulation points and the split points, and mark each edge (or portion of a split edge) with the direction of particle movement along that edge, as in Figure 3. We call a diagram of this a *flow diagram for w with respect to P*.

▶ **Lemma 1.** *In a traversal of ∂P, accumulation and split points alternate.*

Proof. Note that in a flow diagram the only points of the boundary with two opposing directions of particle movement are the accumulation points, where the movement is towards the point, and the split points, where the movement is away from the point. Thus, between any two consecutive split points on the boundary, there must be an accumulation point, and between any two consecutive accumulation points, there must be a split point. This implies the lemma. ◀

▶ **Theorem 2.** *A convex polygon P is 1-gatherable from w iff w lies in the slab of exactly one of the edges of P.*

Proof. A polygon is 1-gatherable from w iff an actuator at w has one accumulation point. Since accumulation and split points alternate, this holds iff the actuator has exactly one split point. Since an actuator has a single split point in every slab that it is in (and no others), the result follows. ◀

The boundary of the slab for edge e consists of e and two rays perpendicular to e. If we produce these two rays for each edge of P, and intersect all these rays with P, we get a set of at most $2n$ chords that define a decomposition that we call the *slab decomposition* of P. An example slab decomposition is shown in Figure 5. The cells of this decomposition have the property that if two points are in a cell, then these two points are in exactly the same set of slabs of P.

Theorem 2 then immediately implies that the repulsion kernel of P is the union of zero or more cells of the slab decomposition of P. This gives us the basis for an $O(n^2)$ time algorithm for finding the repulsion kernel. We start by constructing the slab decomposition. We can use topological sweep to compute a quad-edge data structure for the slab decomposition in $O(n^2)$ time [5, 4, 6].

▶ **Theorem 3.** *The repulsion kernel of a convex polygon can be computed in $O(n^2)$ time.*

Proof. We construct the slab decomposition. As we construct the decomposition, we augment each edge with information about which slab or slabs it borders and to which side of the edge said slabs are on. (An edge may border two slabs if the two slabs each have a defining ray that are collinear.). Choose an arbitrary cell c of the decomposition and determine how many slabs it is in. From this cell, perform a graph search on the dual of the decomposition. Each time we step over an edge, from one cell to another, during this search, we update in constant time the number slabs we are in, according to the information on the edge. We maintain a list of all cells where this value is one. At the end of the search, this list is the repulsion kernel. ◀

If we allow actuators to be located outside a polygon P, then every convex polygon is 1-gatherable.

▶ **Lemma 4.** *Every convex polygon is 1-gatherable from some point in the plane.*

Proof. If you go far enough away, you can always find a point that is not covered by any slab. For this point, there is only one accumulation point. Therefore, an activation of an actuator from this point moves all the particles to the accumulation point. ◀

Given the above, one may be tempted to believe that *every* convex polygon is 1-gatherable when the actuators are restricted to be inside the polygon. However, this is not always the case.

▶ **Lemma 5.** *For $k \geq 2$, the regular $(2k+1)$-gon P_{2k+1} is not 1-gatherable.*

Proof. Assume that the edge length of P_{2k+1} is 2, and that e_0 is oriented with direction 0 (horizontal on the bottom of the polygon). This is illustrated in Figure 4 for P_5.

By Lemma 12, we need only show that P_{2k+1} is not 1-gatherable from its boundary. By symmetry, we need consider only e_{k+1}. The edge e_{k+1} starts at the top center of the polygon and proceeds downward to the left. The slab $S(e_0)$ contains the upper half of e_{k+1}, as the

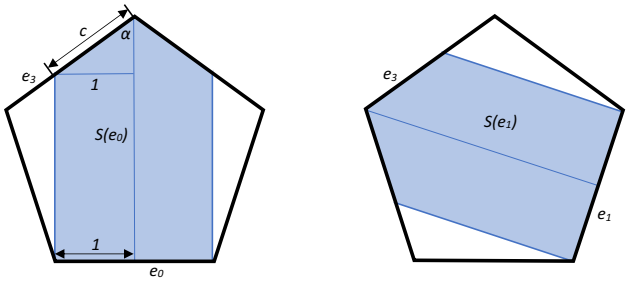


Figure 4 (a) $S(e_0)$ covers the top half of e_3. (b) $S(e_1)$ covers the bottom half.

distance c (see figure) is greater than 1. (It is $1/\sin\alpha$, to be precise, where α is half the vertex angle, or $\frac{(2k-1)\pi}{4k+2}$.) Similarly, the slab $S(e_1)$ contains the bottom half of e_{k+1}.

Thus, each point of e_{k+1} is in $S(e_{k+1})$ and either $S(e_0)$ or $S(e_1)$ or both. Thus, by Theorem 2, the polygon is not 1-gatherable from any point of e_{k+1}.

The vertices are sometimes special cases, but here the vertex v_{k+1} (the top vertex of the polygon) is in $S(e_0)$, $S(e_k)$, and $S(e_{k+1})$, and thus the polygon is not 1-gatherable from there. By symmetry, it is not 1-gatherable from any vertex. ◀

In fact, some convex polygons may be ungatherable. It turns out that acute angles are a major impediment to gathering.

▶ **Lemma 6.** *A particle that is at an acute vertex v of P cannot be moved by an actuator activated at any point in $P \setminus v$.*

Proof. Given any point $p \in P \setminus v$, the acute vertex v is a local maximum with respect to distance since any point in P that is infinitesimially close to v is closer to p than v. ◀

This immediately implies the following.

▶ **Theorem 7.** *A convex polygon with three acute vertices is not k-gatherable for any $k > 0$.*

For the remainder of the paper, we only consider convex polygons with at most two acute vertices.

3 1-Gatherability

We have shown so far that not all convex polygons are 1-gatherable. We have also given a complete characterization of when a convex polygon is 1-gatherable by computing the repulsion kernel of a polygon in $O(n^2)$ time. This begs the question whether it is possible to find a point from which the polygon is 1-gatherable more efficiently, without having to compute the repulsion kernel. We answer this question in the affirmative by providing an $O(n)$ time algorithm. Before presenting the algorithm, we highlight some useful geometric properties.

▶ **Lemma 8.** *Let a be an accumulation point of an actuator activated at w in P. The line L that goes through a and is perpendicular to wa is a line of support of the polygon.*

Proof. Since a is an accumulation point, it is a local maximum of distance from w. Thus, the circle C with center w and radius aw encloses the polygon in the neighborhood of a. The line L is tangent to (outside of) C at a and thus locally supports the polygon at a. Since the polygon is convex, L also globally supports the polygon. ◀

We now show that we can restrict our attention to particles starting only on the boundary of P.

▶ **Lemma 9.** *An actuator in P that 1-gathers all the particles on ∂P also 1-gathers all particles in P.*

Proof. The activation of an actuator in P forces a particle p in the interior of P to move directly away from the actuator until it hits the boundary at some point b. Since there was a particle p' whose initial position is b, the particle p will follow the path of p' and stop at the same place p' stops. Thus, the location of p will always be accounted for by the position of p'. In other words, p is redundant and can be removed from the problem. ◀

We can take this a step further and show that particles located on the interior of edges are redundant.

▶ **Lemma 10.** *An actuator in P that 1-gathers all the particles on the vertices P also 1-gathers all particles on ∂P.*

Proof. The activation of an actuator in P forces a particle p in the interior of an edge of P to move along along the edge until it reaches a vertex v. There was a particle p' that started at v, and we can follow the proof of Lemma 9. ◀

The above lemmas show that particle movement can be restricted to the boundary. In fact, to solve the general problem, we only need to consider the problem where particles are only on vertices. We show a relationship between self-approaching paths and the path on the boundary followed by a particle under the influence of an actuator. Recall that a directed path Π is self-approaching if for any three consecutive points x, y, z on the path, we have the property that $|xz| \geq |yz|$ [7].

▶ **Lemma 11.** *If $\partial P(x, y)$ is self-approaching from x to y then activating an actuator at y sends all the particles on $\partial P(x, y)$ to x along the boundary.*

Proof. Let z be an arbitrary point on $\partial P(x, y)$. We observed that activating an actuator at y will move z along the boundary. We need to establish in which direction the particle will move. Since $\partial P(x, y)$ is self-approaching from x to y, we have that $|yz| \leq |yx|$. Therefore, the particle z will move to x since particles move in a direction to increase their distance from an actuator. ◀

Next, we show that if the repulsion kernel is not empty, then there is at least one point on the boundary that is in the repulsion kernel.

▶ **Lemma 12.** *Let P be a convex polygon that is 1-gatherable from a point w in the interior of P, and let a be the accumulation point for w. Let R be the ray from a through w, not including the point a. Then P is 1-gatherable from the point $w' = R \cap \partial P$, with a as its accumulation point (see Figure 5).*

Proof. By Theorem 2, the point of gatherability w is in one edge e's perpendicular slab. Without loss of generality, we assume that e is horizontal at or below w (by rotation), that a is not to the right of w (by reflection), and that e is $e_0 = v_0v_1$(by labelling). Let m be such that $a = v_m$. See Figure 6. Let p be the point on e which has a perpendicular through w. Note that p is a split point for w.

To show that P is 1-gatherable from the point w' on ∂P, by Lemma 10, it suffices to show that the particles located on the vertices of P move to one accumulation point with

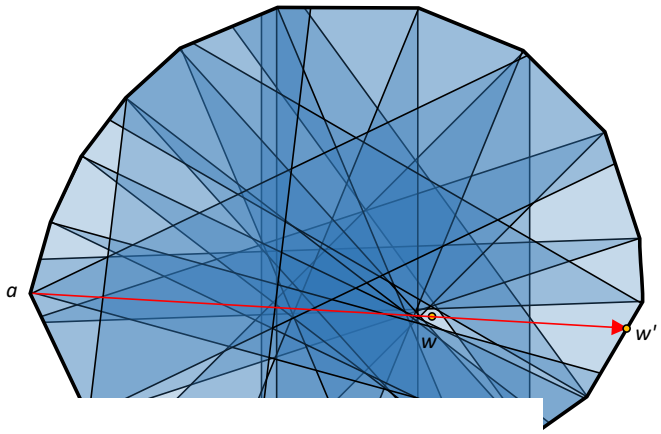


Figure 5 *F* ponding to the number

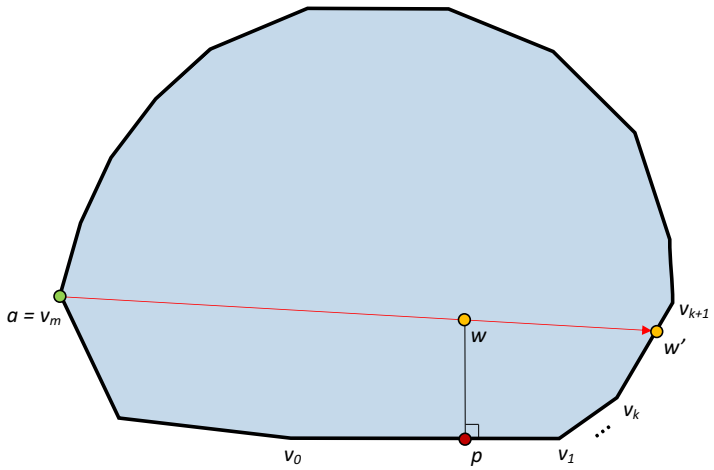


Figure 6 Some relevant points on the polygon.

the activation of an actuator at w'. We will show that this accumulation point is a. We assume without loss of generality that w' is located on the edge $e_k = [v_k v_{k+1})$. Recall that if w' happens to be on v_k, then the placement of the actuator on w' means the particle located at w' is removed from consideration.

We begin with the claim that an accumulation point for w' is a. If this were not the case, then there would be a way to increase the distance from w' on the boundary in the neighborhood of a. By Lemma 8, there is a line L perpendicular to wa that is a line of support of P at a. By construction, L is perpendicular to $w'a$. Therefore, a is a local maximum with respect to w', and thus is an accumulation point for w'. We will now show that particles located at all other vertices move to a when an actuator is activated at w'.

Since p is a split point for w, we have that upon activation of w, the particles on the vertices on $\partial P(a, p)$ move clockwise along the boundary to a. Similarly, the particles on the vertices on $\partial P(p, a)$ move counterclockwise along the boundary to a. By Theorem 2, this means that w is in all of the regions $R_{ccw}(e_1)$, $R_{ccw}(e_1), \ldots, R_{ccw}(e_{m-1})$ and w is in $R_{cw}(e_m)$, $R_{cw}(e_{m+1}), \ldots, R_{cw}(e_{n-1})$. See Figure 8.

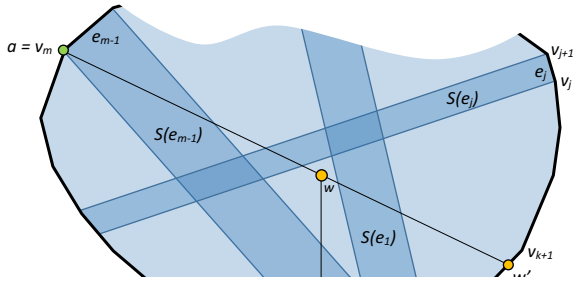


Figure 7 w is

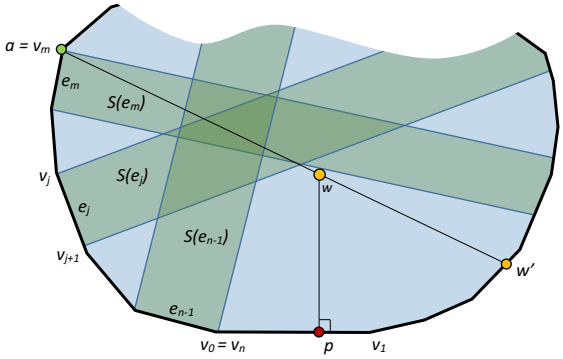


Figure 8 w is in the regions $R_{\text{cw}}(e_m)$ to $R_{\text{cw}}(e_n)$.

Since all of the slabs $S(e_m), S(e_{m+1}), \ldots, S(e_{n-1})$ cross the chord aw' between a and w, we have that w' is also in $R_{\text{cw}}(e_m)$, $R_{\text{cw}}(e_{m+1}), \ldots, R_{\text{cw}}(e_{n-1})$. Thus, the vertices $v_{m+1}, \ldots, v_n$ move in a clockwise direction to a.

Now, we must show that the particles on vertices $v_1, \ldots, v_{m-1}$ also move to a. We first consider the vertices $v_{k+1}, \ldots, v_{m-1}$. Again, since these vertices move counterclockwise when the actuator is activated at w, the slabs $S(e_j)$ for $k \leq j \leq m-1$ cross the chord aw' between a and w. Therefore, none of them can contain w'. This implies that w' is in $R_{\text{ccw}}(e_{k+1}), \ldots, R_{\text{ccw}}(e_{m-1})$.

We now show that the vertices $v_1, \ldots, v_k$ move in a clockwise direction to a. Consider the circle C centered at w and going through w'. This circle contains $\partial P[v_1, v_k]$ since particles on v_1, $v_2, \ldots, v_k$ move in a counterclockwise direction to a when an actuator is activated at w. It is strict containment as the particles always move away from w. Now consider the circle C' that has the chord aw' as diameter. Since a is the accumulation point for w, it is the farthest point from w. This implies that the the center c of C' lies on the segment aw, with radius $|cw'|$. C' contains C since $|cw'| > |ww'|$. (Figure 9).

Let q be an arbitrary point in $\partial P(p, w')$. Since q is in the interior of C, we have that $\angle w'qa > \pi/2$. By convexity, we have that $\angle w'qp > \angle w'qa$. Consider the cone formed by the ray from q to w' and the ray at q that is an extension of the line through a and q. Since $\angle w'qp > \pi/2$, we have that the angle formed at this ray is strictly less than $\pi/2$ and $\partial P[q, w')$ is contained in the cone. Lemma 3 in [7] states that when $\partial P[q, w')$ is contained in a cone

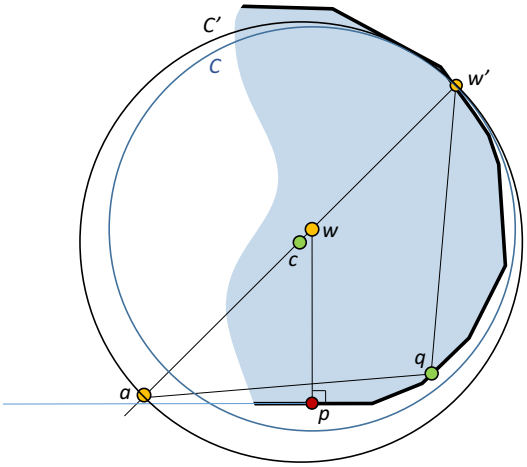


Figure 9 The circle C' contains the circle C and thus contains the boundary from p to w'.

at q with angle at most $\pi/2$ for every $q \in \partial P(p, w')$ then $\partial P(p, w')$ is self-approaching from p to w'. By Lemma 11, we have that an activation of an actuator at w' sends q clockwise around the boundary to p since $|pw'| \geq |qw'|$. Therefore, the vertices $v_1, \ldots, v_k$ move in a clockwise direction to a.

We have now shown the polygon is 1-gatherable from w'. ◀

As a consequence of the previous lemma, in order to tell if a polygon is 1-gatherable, it suffices to determine if it is 1-gatherable from the boundary. To do this in linear time, we employ an approach that resembles the rotating calipers algorithm to compute the diameter of a convex polygon [13]. In essence, for every point x on ∂P, we want to compute the first clockwise and first counterclockwise accumulation point. We do this in two steps. We compute all the counterclockwise accumulation points then compute the clockwise accumulation points. The algorithm to compute the counterclockwise accumulation points proceeds as follows. We start at the lowest point x of P and place the first horizontal caliper at x. We then walk around the boundary in counterclockwise direction until we find the counterclockwise accumulation point y for x. We place the second caliper at y such that it is perpendicular to xy. As x moves counterclockwise around P, there are two types of events. Either x moves to a new vertex or the caliper at y becomes coincident to an edge of P in which case y moves from one vertex to the next. There are a linear number of events that occur and by recording these events, when the calipers returns to its starting positions, we know the counterclockwise accumulation point for every point on the boundary of P. By repeating this in the clockwise direction, we find the clockwise accumulation points. For any point on the boundary of P, if its clockwise accumulation point is the same as its counterclockwise accumulation point, then the polygon is 1-gatherable from that point. We conclude this section with the following:

▶ **Theorem 13.** *We can determine if a convex n-vertex polygon is 1-gatherable in $O(n)$ time.*

Proof. Follows from Lemma 12 and the discussion above. ◀

4 2-Gatherability

In this section we prove that a convex polygon with at most two acute vertices is 2-gatherable. We then give an $O(n)$ algorithm to determine the location of the two actuators and the sequence of activation.

▶ **Theorem 14.** *If a convex polygon has two or fewer acute vertices, then it is 2-gatherable.*

Proof. Let $D(P)$ be the smallest disk enclosing polygon P with centre c. Either there are two vertices v_i and v_j of P that form a diameter of $D(P)$ or there are three vertices v_i, v_j, and v_k on $\partial D(P)$ such that c is in the interior of the triangle formed by the three vertices [12, 15]. We consider each case separately. Recall that by Lemma 9, we can assume that the particles are only located on the boundary of P.

Case 1: Two vertices v_i and v_j of P form a diameter of $D(P)$. In this case, we show that an actuator activated at vertex v_i results in all particles accumulating at v_j. Assume, without loss of generality, that v_i and v_j lie on a vertical line L with v_i below v_j. The two vertices partition the polygon boundary into two chains, $\partial P[v_i, v_j]$ which is to the right of L and $\partial P[v_j, v_i]$ which is to the left. We also assume that each chain consists of at least two edges, since otherwise, one of the chains is the edge v_iv_j and trivially any particle on this edge moves to v_j when an actuator at v_i is activated. To complete the proof in this case, by Lemma 11, it suffices to show that both $\partial P[v_i, v_j]$ and $\partial P[v_j, v_i]$ are self-approaching curves from v_j to v_i.

Consider any point $x \in \partial P(v_i, v_j)$. Since x is in $D(P)$ strictly to the right of L we have that $\pi > \angle v_j x v_i \geq \pi/2$. Consider the cone formed by the intersection of the half-space bounded by the line through v_j and x that contains v_i and the half-space bounded by the line through v_i and x that does not contain v_j. This cone has angle at most $\pi/2$ and contains $\partial P[v_i, x]$. Since x is an arbitrary point on $\partial P(v_i, v_j)$, by Lemma 3 in [7], we have that $\partial P[v_i, v_j]$ is self-approaching from v_j to v_i. A similar argument shows that $\partial P[v_j, v_i]$ is also self-approaching from v_j to v_i.

Case 2: There are three vertices v_i, v_j, and v_k appearing in counter-clockwise order on $\partial D(P)$ such that c is in the interior of the triangle formed by the three vertices. Since there are at most two acute vertices, without loss of generality, assume that v_j is a polygon vertex with interior angle at least $\pi/2$. Reorient the polygon such that v_i is the lowest point. The polygonal chains $\partial P[v_i, v_j]$, $\partial P[v_j, v_k]$ and $\partial P[v_k, v_i]$ are self-approaching from v_j to v_i, v_k to v_j and v_k to v_i, respectively, by the same argument as the one used in Case 1. In fact, since c is strictly in the interior of the triangle formed by the three vertices, we have that the cones used to prove that the chains are self-approaching have an angle that is strictly less than $\pi/2$.

By placing a first active actuator on v_i, we have that all the particles on $\partial P(v_i, v_j]$ and all the particles on $\partial P[v_k, v_i)$ move onto $\partial P[v_j, v_k]$. Since $\partial P[v_j, v_k]$ is self-approaching from v_k to v_j, if we activated a second actuator at v_j then all the particles on this chain move to v_j's accumulation point which would complete the proof. However, even though v_j is not acute, it may be the case that v_j is the counterclockwise accumulation point for v_i. This would prevent us from placing an actuator on v_j since after the activation of the first actuator on v_i, particles have accumulated on v_j. Recall that all subsequent placements of actuators must be on points in P that are free of particles. Since for every point x on $\partial P(v_j, v_k)$, $\angle v_j x v_k > \pi/2$, there must exist a point y on the edge v_jv_{j-1} infinitessimally close to v_j such

that the $\angle yzv_k$ is still strictly greather than $\pi/2$ for every $z \in \partial P[v_j, v_k)$. This implies that $\partial P[y, v_k]$ is self-approaching from v_k to y. Thus, by Lemma 11, activating a second actuator at y, which is free of particles after the first activation, moves all the particles that have accumulated on $\partial P[v_j, v_k]$ to the counterclockwise accumulation point of y. ◀

References

1 Sang Won Bae, Chan-Su Shin, and Antoine Vigneron. Improved bounds for beacon-based coverage and routing in simple rectilinear polygons. *arXiv preprint arXiv:1505.05106*, 2015.
2 Michael Biro. *Beacon-based routing and guarding.* PhD thesis, State University of New York at Stony Brook, 2013.
3 Michael Biro, Justin Iwerks, Irina Kostitsyna, and Joseph SB Mitchell. Beacon-based algorithms for geometric routing. In *WADS*, pages 158–169. 2013.
4 Herbert Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38(1):165–194, 1989.
5 Herbert Edelsbrunner and Leonidas J. Guibas. Corrigendum: Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 42(2):249–251, 1991.
6 Leonidas J. Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and computation of voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, 1985.
7 Christian Icking, Rolf Klein, and Elmar Langetepe. Self-approaching curves. *Math. Proc. Camb. Phil. Soc.*, 123(3):441–453, 1999.
8 Irina Kostitsyna, Bahram Kouhestani, Stefan Langerman, and David Rappaport. An optimal algorithm to compute the inverse beacon attraction region. In *SoCG*, 2018.
9 Bahram Kouhestani, David Rappaport, and Kai Salomaa. On the inverse beacon attraction region of a point. In *CCCG*, 2015.
10 Bahram Kouhestani, David Rappaport, and Kai Salomaa. The length of the beacon attraction trajectory. In *CCCG*, pages 69–74, 2016.
11 Bahram Kouhestani, David Rappaport, and Kai Salomaa. Routing in a polygonal terrain with the shortest beacon watchtower. *Comput. Geom.*, 68:34–47, 2018.
12 Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.
13 Michael Shamos. *Computational Geometry.* PhD thesis, Yale Univeristy, 1978.
14 Thomas C Shermer. A combinatorial bound for beacon-based routing in orthogonal polygons. *arXiv preprint arXiv:1507.03509*, 2015.
15 Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, pages 359–370, 1991.

Improved Bounds for Guarding Plane Graphs with Edges

Ahmad Biniaz[1]
Cheriton School of Computer Science, University of Waterloo
Waterloo, Canada
ahmad.biniaz@gmail.com

Prosenjit Bose[2]
School of Computer Science, Carleton University
Ottawa, Canada
jit@scs.carleton.ca

Aurélien Ooms[3]
Département d'Informatique, Université libre de Bruxelles (ULB)
Brussels, Belgium
aureooms@ulb.ac.be

Sander Verdonschot[4]
School of Computer Science, Carleton University
Ottawa, Canada
sander@cg.scs.carleton.ca

Abstract

An *edge guard set* of a plane graph G is a subset Γ of edges of G such that each face of G is incident to an endpoint of an edge in Γ. Such a set is said to *guard* G. We improve the known upper bounds on the number of edges required to guard any n-vertex embedded planar graph G:

1. We present a simple inductive proof for a theorem of Everett and Rivera-Campo (1997) that G can be guarded with at most $\frac{2n}{5}$ edges, then extend this approach with a deeper analysis to yield an improved bound of $\frac{3n}{8}$ edges for any plane graph.
2. We prove that there exists an edge guard set of G with at most $\frac{n}{3} + \frac{\alpha}{9}$ edges, where α is the number of quadrilateral faces in G. This improves the previous bound of $\frac{n}{3} + \alpha$ by Bose, Kirkpatrick, and Li (2003). Moreover, if there is no short path between any two quadrilateral faces in G, we show that $\frac{n}{3}$ edges suffice, removing the dependence on α.

2012 ACM Subject Classification Theory of computation → Computational geometry, Mathematics of computing → Graph theory

Keywords and phrases edge guards, graph coloring, four-color theorem

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.14

1 Introduction

The original Art Gallery Problem: "How many guards are necessary, and how many are sufficient to patrol the paintings and works of art in an art gallery with n walls?" was posed by Victor Klee in 1973. Chvatal [5] offered the first solution to the question by

[1] Supported by NSERC and Fields postdoctoral fellowships.
[2] Supported by NSERC.
[3] Supported by the Fund for Research Training in Industry and Agriculture (FRIA).
[4] Partially supported by NSERC and the Carleton-Fields Postdoctoral Award.

proving that $n/3$ guards are sufficient and sometimes necessary to guard an n-vertex polygon. However, since then, an active area of research was spawned, where researchers studied many different variants of the problem, by allowing different types of guards and guarding different types of objects. The field is vast and many surveys on the topic have been written (see [11, 13, 10, 12]). In this paper, the variant we study is when the guards are edges and the object guarded is a plane graph.

A *plane graph* is a graph that is embedded in the plane without crossing edges. Throughout this paper, G is a plane graph with $n \geq 3$ vertices and at least one edge. The graph G divides the plane into regions called the *faces* of G. A *guard set* for G is a subset Γ of edges of G such that every face of G (including the outer face) contains at least one endpoint of an edge in Γ on its boundary. In other words, when the endpoints of the edges of Γ guard the faces of G, we say that Γ *guards* G. We focus on the problem of finding a guard set for G with minimum size. To avoid some notational clutter, we omit floors and ceilings in the statements of the bounds. However, since the size is necessarily integer, all fractional bounds can be rounded down for upper bounds and rounded up for lower bounds, except in the case when the upper bound is less than 1, in which case, we round up to 1.

For maximal outerplanar graphs, O'Rourke [9] showed that $\frac{n}{4}$ edge guards are always sufficient and sometimes necessary. In his proof, both the upper bound and lower bound require that every bounded face is a triangle and the outer face is a cycle. By removing this restriction, both the upper and lower bounds jump to $\frac{n}{3}$ for arbitrary outerplanar graphs [4, 5]. For maximal plane graphs (triangulations), Everett and Rivera-Campo [6] showed that $\frac{n}{3}$ edge guards are always sufficient and Bose et al. [4] showed that $\frac{4n-4}{13}$ edge guards are sometimes necessary. The upper bound is derived using the four-color theorem. Note the gap between the upper and lower bounds. The lower bound is derived by constructing a triangulation where $\frac{4n-4}{13}$ triangles are *isolated*. Two triangles are isolated if there is no edge joining a vertex of one triangle with a vertex of the other triangle. Since it is impossible to isolate $\frac{n}{3}$ triangles in a maximal plane graph, this would suggest that the upper bound argument may not be exploiting all of the structure present in a maximal plane graph.

Indeed, when one studies plane graphs that are no longer restricted to be maximal, the current best upper bound is no longer $\frac{n}{3}$. Everett and Rivera-Campo [6] used the four-color theorem to prove that $\frac{2n}{5}$ edges suffice. By using a different coloring approach, Bose, Kirkpatrick and Li [3] proved that $\frac{n}{3} + \alpha$ edges are sufficient, where α is the number of quadrilateral faces of G. Since outerplanar graphs are planar, $\frac{n}{3}$ edges are sometimes necessary and no better lower bound is known. Although it seems that the number of quadrilateral faces plays a key role in this problem, it is unclear which upper bound is better in the worst case: $\frac{2n}{5}$ or $\frac{n}{3} + \alpha$, since α can be as high as $n - 2$. Our main contribution is an improvement on both upper bounds. We give a simpler proof for Everett and Rivera-Campo's upper bound of $\frac{2n}{5}$ edges. In addition, by exploiting various properties of planar graphs, we are able to strengthen the bound to $\frac{3n}{8}$ edges. We then show that, for plane graphs with α quadrilateral faces, $\frac{n}{3} + \frac{\alpha}{9}$ edges suffice, reducing the dependency on α. Table 1 summarizes the best known upper and lower bounds.

2 Iterative Guarding

We first introduce a proof strategy that iteratively builds a guard set while shrinking the graph. We use this strategy to give a simple proof of Everett and Rivera-Campo's [6] result that $\frac{2n}{5}$ edges suffice for any plane graph, before strengthening this bound to $\frac{3n}{8}$. Note that, if the graph has a single face, it can be guarded by one edge and our bounds hold so long as $n \geq 3$. In the remainder of this section, we assume that the initial graph has at least two faces.

Table 1 The best known upper and lower bounds for various types of graphs, where n is the number of vertices and α is the number of quadrilateral faces.

Graph Type	Lower Bound	Upper Bound
Maximal Outerplanar	$\frac{n}{4}$ [9]	$\frac{n}{4}$ [9]
Outerplanar	$\frac{n}{3}$ [4]	$\frac{n}{3}$ [5]
Maximal Planar	$\frac{4n-4}{13}$ [4]	$\frac{n}{3}$ [6]
Planar	$\frac{n}{3}$ [4]	$\min\{\frac{n}{3}+\frac{\alpha}{9}, \frac{3n}{8}\}$ [this paper]

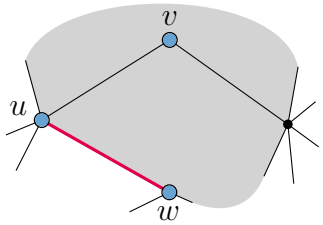


Figure 1 Edge (u, w) guards both faces incident to a vertex v of degree 2, allowing us to remove all three vertices.

The general strategy works as follows. Suppose we are aiming for a bound of cn edges, for some constant $c > 0$. We start with an empty partial guard set $\Gamma = \emptyset$. Given a plane graph G, we identify a set of vertices V' and edges E' such that (i) the edges in E' guard all faces incident to vertices in V' and (ii) we have that $|E'| \leq c|V'|$. We then add all edges of E' to Γ, remove all vertices in V' from G, along with their incident edges, and repeat until G has one face left; i.e. G is a forest. This face has already been guarded in the penultimate step, so we return Γ as our guard set. Since we added at most c edges for every vertex we removed, its size is at most cn.

As a warmp-up, we use this strategy to prove the following bound for 2-degenerate graphs (an undirected graph is k-degenerate if every subgraph has a vertex of degree at most k).

▶ **Theorem 1.** *Every 2-degenerate plane graph with $n \geq 3$ vertices can be guarded by at most $\frac{n}{3}$ edges.*

Proof. Let G be a 2-degenerate plane graph with $n \geq 3$ vertices. If G has one face, we guard it with a single edge and the theorem holds, so assume that G has more than one face. We use the iterative strategy described above to construct a guard set Γ for G with $c = \frac{1}{3}$. Thus, all that is left to do is to describe how to find the sets E' and V'.

We consider two cases, depending on the minimum degree of G. If G contains any vertex v of degree 0 or 1, we let $E' = \emptyset$ and $V' = \{v\}$. While this does not technically satisfy our definition above that the edges in E' guard all faces incident to vertices in V', this operation is still safe, since any guard set for $G \setminus \{v\}$ is also a guard set for G.

If G does not contain any vertex of degree 0 or 1, the fact that it is 2-degenerate tells us that it must have a vertex v of degree 2. Let u be a neighbor of v, and let $w \neq v$ be another neighbor of u (see Figure 1). Such a vertex w must exist, since G has minimum degree 2. We now let $E' = \{(u, w)\}$ and $V' = \{v, u, w\}$. Since edge (u, w) guards both faces incident to v, as well as all faces incident to u and w, this completes the proof. ◀

This gives an alternate proof for the bound on outerplanar graphs [5, 7], since they are 2-degenerate.

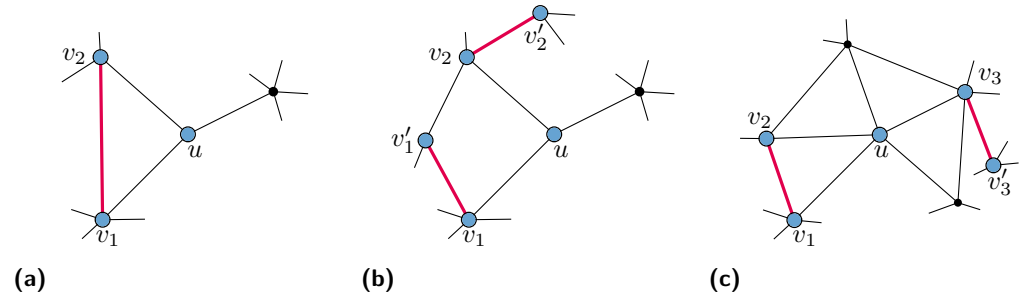


Figure 2 Guarding a vertex (a) of degree 3 with two neighbors connected by an edge; (b) of degree 3 with no neighbors connected by an edge; (c) of degree 4 or 5 and incident to a triangle.

▶ **Corollary 2.** *Every embedded outerplanar graph with $n \geq 3$ vertices can be guarded by at most $\frac{n}{3}$ edges.*

Since a set of $\frac{n}{3}$ disjoint triangles comprises an outerplanar and 2-degenerate graph, the bounds of Theorem 1 and Corollary 2 are best possible for these classes.

We use the same technique to prove the $\frac{2n}{5}$ and $\frac{3n}{8}$ bounds. Since $\frac{1}{3} < \frac{3}{8} < \frac{2}{5}$, we can use the arguments from the proof of Theorem 1 to eliminate vertices of degree 2 or less, even if we are shooting for $c = \frac{2}{5}$ or $c = \frac{3}{8}$. Thus, we may assume for the remainder of the section that the graph has minimum degree 3. Since planar graphs are 5-degenerate, we still need to handle vertices of degree 3, 4, or 5. The following lemma gives us a little more to work with in these cases. For brevity, we denote a vertex of degree d as a d-vertex, and one with degree at most d as a d^--vertex. Likewise, we denote a face with k boundary edges as a k-face and one with at most k edges as a k^--face.

▶ **Lemma 3** (Lebesgue [8])**.** *In each plane graph with minimum degree 3 there exists either a 3-vertex incident to a 5^--face, or a 4-vertex incident to a 3-face, or a 5-vertex incident to four 3-faces.*

▶ **Theorem 4.** *Every plane graph with $n \geq 3$ vertices can be guarded by at most $\frac{2n}{5}$ edges.*

Proof. We use the iterative method with $c = \frac{2}{5}$ and, as argued above, can assume that our graph G has minimum degree at least 3.

First consider the case where G has a vertex u of degree 3. Any two neighbors of u together are incident to all faces incident to u. If any two neighbors v_1 and v_2 of u are connected by an edge, we let $E' = \{(v_1, v_2)\}$ and $V' = \{u, v_1, v_2\}$ (see Figure 2a). Otherwise, let v_1 and v_2 be any two neighbors of u, and let $v_1' \neq u$ be a neighbor of v_1 and $v_2' \notin \{u, v_1'\}$ a neighbor of v_2 (see Figure 2b). We set $E' = \{(v_1, v_1'), (v_2, v_2')\}$ and $V' = \{u, v_1, v_1', v_2, v_2'\}$.

Now suppose that G has minimum degree at least 4. Then Lemma 3 tells us that there must be a 5^--vertex u incident to a triangle. Let v_1 and v_2 be the other vertices of this triangle. Edge (v_1, v_2) guards three of the four or five faces incident to u. Let v_3 be a neighbor of u incident to the faces not guarded by (v_1, v_2), and let $v_3' \notin \{u, v_1, v_2\}$ be a neighbor of v_3 (see Figure 2c). We set $E' = \{(v_1, v_2), (v_3, v_3')\}$ and $V' = \{u, v_1, v_2, v_3, v_3'\}$ (see Figure 2b).

Thus, in each case we can find E' and V' such that the edges of E' guard all faces incident to vertices in V' and $|E'| \leq \frac{2}{5}|V'|$. ◀

To improve this bound further to $\frac{3n}{8}$, we need an even stronger version of Lemma 3, inspired by Borodin [2]. Following his terminology, an edge is incident on a face if one of its

endpoints is on the face. An edge is *weak* if it is incident to two triangles, *semiweak* if it is incident to exactly one triangle, and *strong* otherwise.

▶ **Lemma 5.** *Every plane graph with minimum degree* 3 *contains one of the following:*
(L_1) *a weak edge joining a* 3*-vertex to a* 10^-*-vertex;*
(L_2') *a weak edge joining a* 4*-vertex to a* 6^-*-vertex;*
(L_2'') *a weak edge joining a* 4*-vertex* u *to a* 7*-vertex* v *such that at least one edge adjacent to* (u, v) *around* v *is weak;*
(L_3) *a weak edge joining a* 5*-vertex incident to at least four* 3*-faces to a* 6^-*-vertex;*
(L_4) *a semiweak edge joining a* 3*-vertex to an* 8^-*-vertex;*
(L_5) *a semiweak edge joining a* 4*-vertex to a* 5^-*-vertex;*
(L_6) *an edge incident to a* 4*-face and joining a* 3*-vertex to a* 5^-*-vertex;*
(L_7) *a* 5*-face incident to at least four* 3*-vertices.*

Proof. Borodin [2] proved this lemma, except with configurations (L_2') and (L_2'') replaced by (L_2): a weak edge joining a 4-vertex to a 7^--vertex. We describe how to adapt Borodin's discharging argument to prove our stronger version. For full details, see the original paper [2].

Initially, we assign a charge of $d - 4$ to each d-vertex and each d-face. By Euler's formula, this results in a total charge of -8. Then, following Borodin, we redistribute the charge as follows:

- Every face with more than 4 sides transfers $\frac{1}{3}$ to every 3-vertex on its boundary.
- Every vertex transfers $\frac{1}{3}$ to each incident triangle.
- Each vertex u transfers the following to the other endpoint v of each incident edge:
  - $\frac{2}{3}$ if v has degree 3 and (u, v) is weak;
  - $\frac{1}{2}$ if v has degree 3 and (u, v) is semiweak;
  - $\frac{1}{3}$ if v has degree 3 and (u, v) is strong and u has degree at least 6;
  - $\frac{1}{3}$ if v has degree 4 and (u, v) is weak;
  - $\frac{1}{6}$ if v has degree 4 and (u, v) is semiweak;
  - $\frac{1}{6}$ if v has degree 5 and (u, v) is weak and v is incident to four triangles.

We now assume that G does not contain any of the configurations (L_1) through (L_7), and show that this implies that every vertex and face has non-negative charge – a contradiction. The only change from the original proof is that we cannot assume that weak edges between 4-vertices and 7-vertices do not exist. This only affects the part of the proof dealing with 7-vertices, so if we can show that 7-vertices still have non-negative charge, we are done.

Consider any 7-vertex u. Initially, u has charge $+3$. If there is no weak edge connecting u to a 4-vertex, the original proof still applies, so suppose that v is a neighboring 4-vertex and (u, v) is weak. Then u transfers $\frac{1}{3}$ of its charge to v and each of the two triangles incident to (u, v), leaving it with $+2$ charge. Let v_- and v_+ be the neighbors of u preceding and following v in clockwise order around u, respectively. Since G does not contain configuration (L_2''), neither (u, v_-) nor (u, v_+) is weak, so u does not transfer any charge to the other faces incident to these edges. Furthermore, v_- and v_+ must have degree at least 6, otherwise their edge to v would create configuration (L_2') or (L_5). Therefore they receive no charge from u either.

Even if the remaining faces all receive $\frac{1}{3}$ charge and the remaining vertices $\frac{1}{6}$, this would still leave u with positive charge. By (L_1) and (L_4), no neighbor of u can receive more than $\frac{1}{3}$ charge. If u has another 4-vertex v' as neighbor with (u, v') weak, this results in even less charge distribution, since the neighbors before and after v' do not receive any charge and they cannot overlap with v_+ or v_-, since (u, v_-) and (u, v_+) are not weak. Finally, a 3-vertex connected to u by a strong edge would receive $\frac{1}{3}$ charge, but would prevent the adjacent faces from receiving charge. Thus, u will have non-negative charge after redistribution, which completes the proof. ◀

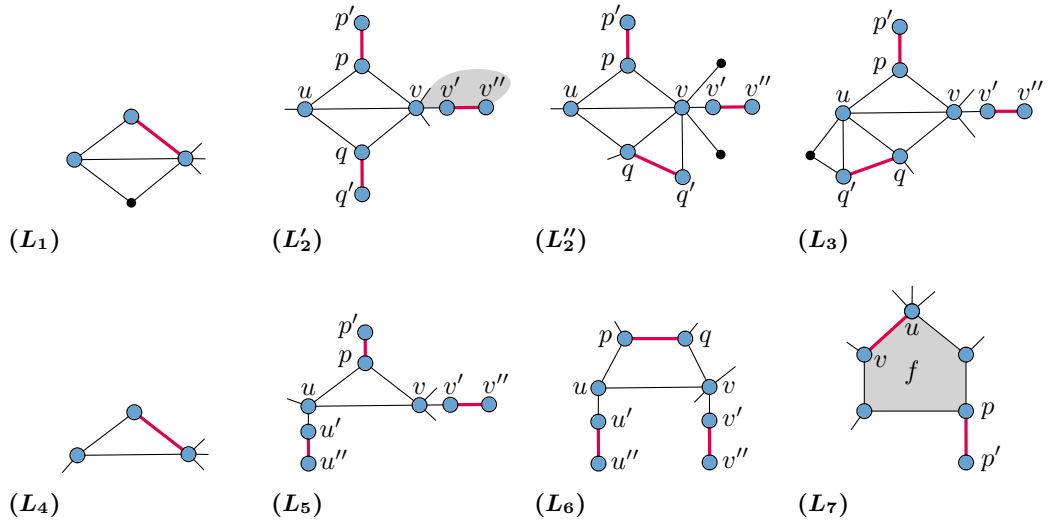


Figure 3 How to select E' (thick shaded edges) and V' (large shaded vertices) in each configuration of Lemma 5.

With Lemma 5 in hand, we can improve our bound to $\frac{3n}{8}$.

▶ **Theorem 6.** *Every plane graph with $n \geq 3$ vertices can be guarded by at most $\frac{3n}{8}$ edges.*

Proof. As before, we use the iterative method and assume that the minimum degree of our plane graph G is 3. We describe how to find E' and V' for each configuration of Lemma 5 (see Figure 3).

If G contains (L_1) or (L_4), we consider a triangle incident to the (semi) weak edge and let E' be the edge of the triangle that is not incident to the 3-vertex. Then V' consists of the 3-vertex and both endpoints of the edge in E'. Thus, for the remainder of the proof, we can assume that any vertex incident to a triangle has degree at least 4.

If G contains (L_2'), let u be its 4-vertex, v be its 6^--vertex, and p and q be the other vertices of the triangles incident to (u, v) (we leave these definitions implicit for the remaining cases; refer to Figure 3). We consider a neighbor p' of p. If p has an edge to q, we let $E' = \{(p, q)\}$ and $V' = \{u, p, q\}$, so suppose that $p' \neq q$. Since q has degree at least 4, it has a neighbor $q' \neq p'$. We add (p, p') and (q, q') to E'. If this guards all faces incident to v, we simply set $V' = \{u, v, p, p', q, q'\}$. Otherwise, let v' be a neighbor of v incident to all unguarded faces (there can be at most two, since v is a 6^--vertex). Let $v'' \neq v$ be the other neighbor of v' along the boundary of one of the unguarded faces incident to v. We know that $v'' \notin \{p, p', q, q', u\}$, otherwise the face would already have been guarded. Thus, we can add (v', v'') to E' and set $V' = \{u, v, p, p', q, q', v', v''\}$.

If G contains (L_2''), we again set $E' = \{(p, q)\}$ with $V' = \{u, p, q\}$ if edge (p, q) exists. Otherwise, let $q' \neq u$ be the other neighbor of q adjacent around v. Since p' has degree at least 4, it has a neighbor $p' \neq q'$. We add (p, p') and (q, q') to E'. If all faces incident to v are guarded, we set $V' = \{u, v, p, p', q, q'\}$. Otherwise, we use the same reasoning as in the previous case to find an extra edge (v', v'') that guards the remaining faces around v.

If G contains (L_3), let $q' \neq v$ be the other neighbor of q adjacent around u. Since p has degree at least 4, it either has a neighbor $p' \notin \{q, q'\}$, or it is connected to both q and q'. In the first case, we add (p, p') and (q, q') to E' and again find a third edge (v', v'') to cover the remaining faces around v. In the second case, v must have a neighbor $v' \neq q'$ otherwise these

five vertices would form a K_5. Then we let $E' = \{(q, q'), (v, v')\}$ and $V' = \{u, p, q, q', v, v'\}$, since (q, q') guards all faces incident to both u and p except for the triangle uvp.

If G contains (L_5), let $u' \neq p$ be the other neighbor of u adjacent to v around u. If p and u' are connected by an edge, let $E' = \{(p, u')\}$ and $V' = \{u, p, u'\}$. Otherwise, let $u'' \notin \{u, v\}$ be a neighbor of u' and let $p' \notin \{u, v, u''\}$ be a neighbor of p. These neighbors exist since u' and p have minimum degree 3 and 4, respectively. We add (u', u'') and (p, p') to E' and, if necessary, find a third edge (v', v'') to cover the remaining faces around v as before. Thus, we get $E' = \{(p, p'), (u', u''), (v', v'')\}$ and $V' = \{u, u', u'', p, p', v, v', v''\}$.

If G contains (L_6), either u is connected to q or it has a neighbor $u' \neq q$. In the first case, we let $E' = \{(p, q)\}$ and $V' = \{u, p, q\}$. In the second case, if u' is connected to any vertex $x \in \{p, q, v\}$ then that edge would cover all faces around u and give us $E' = \{(u', x)\}$ and $V' = \{u, u', x\}$. Otherwise, let $u'' \neq u$ be another neighbor of u'. We add (p, q) and (u', u'') to E' and again find another edge to cover the remaining faces around v.

Finally, if G contains (L_7), let f be the 5-face and let u be a vertex of maximum degree on f. Let v be one of u's neighbors around the face and let p be the vertex on f not adjacent to u or v around f. If p has an edge to u or v, then that edge covers all faces around p, u, and v and we are done. Otherwise, let $p' \notin \{u, v\}$ be a neighbor of p not on f. We set $E' = \{(u, v), (p, p')\}$ and $V' = V(f) \cup \{p'\}$.

Thus, in each case we can find a set E' and V' such that $|E'| \leq \frac{3}{8}|V'|$. ◀

3 Guarding by Coloring

Historically, many questions about guard placement have been resolved by finding an appropriate vertex or edge coloring. Bose et al. [3] defined a *face-respecting k-coloring* of a plane graph G as a k-coloring of the vertices of G such that no face is monochromatic. They were particularly interested in face-respecting 2-colorings with the additional property that every face has a monochromatic edge. For brevity, we call such colorings *guard colorings*. They proved the following result, which we include here as a good introduction to the general technique.

▶ **Lemma 7** (Bose et al. [3]). *If a plane graph with $n \geq 3$ vertices has a guard coloring, it can be guarded by $\frac{n}{3}$ edges.*

Proof. Consider two subgraphs G_1 and G_2 of G, induced by the two color classes of the guard coloring. Let M_1 be a maximal matching in G_1 and M_2 in G_2. Now consider a face f that has a boundary edge e with both endpoints in G_1. Since M_1 is maximal, if it does not contain e, it must contain one of its endpoints. Otherwise, we would obtain a larger matching by adding e. Thus, in each case, M_1 guards f. Recall that one of the properties of a guard coloring is that every face has a monochromatic edge. This implies that $M_1 \cup M_2$ is a guard set for G.

We now have one guard set for G, but we do not have a good bound on the size of this guard set. Indeed, there are examples where $M_1 \cup M_2$ contains many more than $\frac{n}{3}$ edges. To prove the lemma, we find two more guard sets for G such that the total size of all three guard sets is n. Then the smallest of these three sets must have size at most $\frac{n}{3}$.

Our second guard set starts with all edges of M_1, and then adds one edge incident to each vertex of G_1 that is not in M_1. Since our guard coloring has no monochromatic faces, each face has a vertex in G_1. Thus, this set is also a guard set for G. We obtain our third guard set by repeating this construction for M_2.

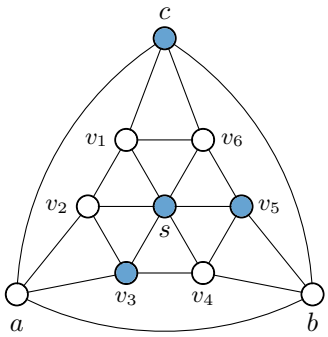


Figure 4 A plane graph without a guard coloring. The illustrated 2-coloring is forced under the assumption that a and b have the same color, but leaves quadrilateral bv_5v_6c without a monochromatic edge.

The size of the first guard set is $|M_1| + |M_2|$. The other guard sets have size $|M_1| + |V(G_1)| - 2|M_1| = |V(G_1)| - |M_1|$, and $|V(G_2)| - |M_2|$, respectively. Thus, in the total size the size of the matchings cancels and we are left with $|V(G_1)| + |V(G_2)| = n$. ◀

Bose et al. also showed that every plane graph without quadrilateral faces has a guard coloring. Thus, a natural question is whether *all* plane graphs – even those with quadrilateral faces – have a guard coloring? In the following theorem we show that this is not the case.

▶ **Theorem 8.** *There are plane graphs that have no guard coloring.*

Proof. Consider the graph in Figure 4. We need to color its vertices with two colors, say white and blue, such that every face contains (i) vertices of both colors and (ii) an edge whose endpoints have the same color. We show that such a coloring does not exist.

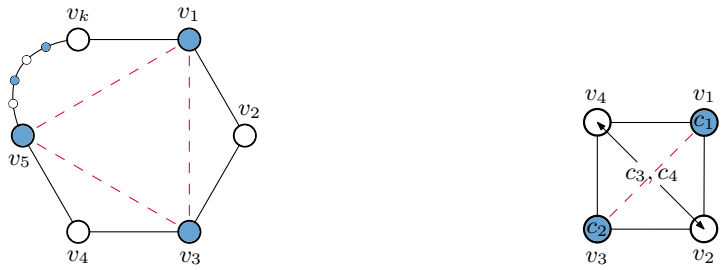
Suppose, for a contradiction, that it does. Since the outer face is a triangle, two of its vertices must have the same color, say white. Suppose that the two vertices are a and b; the other cases are symmetric. This forces c to be blue, since otherwise triangle abc would be monochromatic. Now either v_1 or v_6 needs to be white, otherwise triangle cv_1v_6 is monochromatic. Since the graph is symmetric, we suppose without loss of generality that v_1 is white. This forces v_2 to be white as well, otherwise quadrilateral acv_1v_2 would not have a monochromatic edge. This, in turn, forces s and v_3 to be blue, since they are part of triangles with two white vertices. Now a sequence of such triangles forces v_4 to be white, v_5 blue, and v_6 white. But this leaves quadrilateral bv_5v_6c without a monochromatic edge. Since the entire coloring was forced, this graph has no guard coloring. ◀

Note that this counter-example does not require a large guard set: $\frac{n}{5} = 2$ edges suffice. Thus, it only shows that the technique of guard colorings does not extend to all plane graphs.

Everett and Rivera-Campo [6] used a different vertex coloring to find small guard sets. We modify their approach here to give an upper bound that improves on the $\frac{n}{3} + \alpha$ bound by Bose et al. [3].

▶ **Theorem 9.** *Every plane graph with $n \geq 3$ vertices and α quadrilateral faces can be guarded by at most $\frac{n}{3} + \frac{\alpha}{9}$ edges.*

Proof. We first construct a triangulation G' by inserting extra diagonals in every non-triangular face of G, with two restrictions. First, we do not insert edges that are already in G. Second, for every k-face with $k \geq 6$ and boundary $v_1, v_2, \ldots, v_k, v_1$, we first add the



(a) **(b)**

Figure 5 A triangulation and coloring of the faces of G. The red dashed edges are added when triangulating (a) a face with six or more sides and (b) a quadrilateral.

three edges v_1v_3, v_3v_5, and v_5v_1 (see Figure 5a). By the four-color theorem [1], we can find a proper coloring of G' with a set of four colors $\{c_1, c_2, c_3, c_4\}$. Consider one such coloring, and note that it is also a proper coloring of G.

Since each face of G was triangulated in G', its vertices have at least three distinct colors. Thus, if we consider any two colors, say c_1 and c_2, each face has a vertex with at least one of these two colors. In other words, each face of G contains a vertex of G_{12}, the subgraph of G induced by the vertices with color c_1 or c_2. This means we can create a guard set for G by finding a set of edges whose endpoints include all vertices of G_{12}. We do this by finding a maximal matching M_{12} of G_{12}, then adding one extra edge incident to each vertex of G_{12} not in M_{12}. We call the resulting guard set Γ_{12}, and note that it contains $|\Gamma_{12}| = |V(G_{12})| - |M_{12}|$ edges, since each edge of M_{12} covers two vertices in G_{12}. We can do this for each combination of two colors, giving us six different guard sets.

Now consider the set $\Gamma_{1234} = M_{12} \cup M_{34}$. We show that this is a guard set for all non-quadrilateral faces of G. First, suppose that some face has an edge e whose endpoints have colors c_1 and c_2. If neither endpoint of e is in M_{12}, we can add e to M_{12} to obtain a larger matching. But M_{12} is maximal, so it must already contain some edge incident to an endpoint of e. Thus, M_{12} guards all faces with a (c_1, c_2)-edge. We claim that every non-quadrilateral face of G has either a (c_1, c_2)-edge, or a (c_3, c_4)-edge and is therefore guarded by Γ_{1234}. To show this, we group colors c_1 and c_2 into one color class c_A and c_3 and c_4 into c_B. Our claim is equivalent to saying that every non-quadrilateral face has a monochromatic edge in this two-coloring. This is clear for faces of odd length, since they cannot be properly two-colored.

Let f be a k-face with $k \geq 6$ and with boundary $v_1, \ldots, v_k$ (see Figure 5a). To avoid a monochromatic edge, the colors c_A and c_B must alternate along the boundary. This means that v_1, v_3, and v_5 get the same color. But these form a triangle in G', since we started triangulating this face by inserting the edges v_1v_3, v_3v_5, and v_5v_1. Thus, they must have three distinct colors in the four-coloring, which means they cannot have the same color in the two-coloring. Therefore Γ_{1234} guards all non-quadrilateral faces. An analogous argument shows that the same holds for $\Gamma_{1324} = M_{13} \cup M_{24}$ and $\Gamma_{1423} = M_{14} \cup M_{23}$.

What about quadrilateral faces? Let q be a quadrilateral face with boundary v_1, v_2, v_3, v_4 and suppose that it was triangulated by adding v_1v_3 (see Figure 5b). We show that at least two of Γ_{1234}, Γ_{1324}, and Γ_{1423} guard q. Suppose that q is not guarded by Γ_{1234}, which means that it does not have (c_1, c_2)-edges, or (c_3, c_4)-edges. Without loss of generality, assume that v_1 has color c_1. Then the two-coloring argument and the presence of edge v_1v_3 force v_3 to have color c_2, while v_2 and v_4 have color c_3 or c_4. Either way, there is both a (c_1, c_3)- or

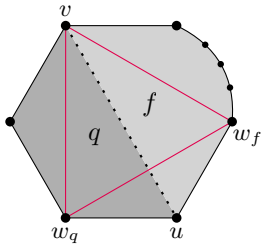


Figure 6 Triangulating the face resulting from merging quadrilateral q with a neighboring face f.

(c_2, c_4)-edge and a (c_1, c_4)- or (c_2, c_3)-edge. By symmetry, this means that if one of the three does not guard q, the other two do. We complete Γ_{1234} to a guard set by adding, for each quadrilateral q not guarded by Γ_{1234}, one edge incident to q, and likewise for Γ_{1324} and Γ_{1423}. The total size of these three guard sets is $|M_{12}| + |M_{34}| + |M_{13}| + |M_{24}| + |M_{14}| + |M_{13}| + \alpha$.

We now have nine guard sets for G. The total number of edges in these sets is $3n + \alpha$, since each vertex occurs in three of the G_{ij}, and the size of the matchings cancels. Thus, the smallest of these sets has size at most $\frac{3n+\alpha}{9} = \frac{n}{3} + \frac{\alpha}{9}$. ◀

4 Distant Quadrilaterals

In this section, we combine both methods used previously to prove a better upper bound for plane graphs in which every pair of quadrilaterals is far apart. To make this more precise, we say that two faces f and g are *h-hop apart* if every path from a vertex on the boundary of f to a vertex on the boundary of g contains at least h edges.

▶ **Theorem 10.** *Every plane graph with $n \geq 3$ vertices in which every two quadrilateral faces are 3-hop apart can be guarded by at most $\frac{n}{3}$ edges.*

Proof. We first use the iterative algorithm as described in the proof of Theorem 1 to remove any vertices of degree less than 3. We have to be a little careful here, since removing these vertices could introduce a new quadrilateral face that is not 3-hop apart from existing quadrilaterals. To remedy this, we first mark all quadrilateral faces in the original graph. Now, if removing a vertex v of degree 1 would introduce a new quadrilateral face, we instead consider its neighbor u and another of u's neighbors $w \neq v$ (these vertices must exist if removing v would introduce a new quadrilateral). We then add (u, w) to our partial guard set Γ_1 and remove all three vertices. This guarantees that all newly introduced quadrilaterals are guarded by Γ_1, since we already do the same for vertices of degree 2.

If the graph was 2-degenerate, we are now done. Otherwise, this results in a graph G with minimum degree at least 3 and a partial guard set Γ_1 of size at most $\frac{n_1}{3}$, where n_1 is the number of vertices removed. We proceed to find a guard set Γ_2 for G of size at most $\frac{n_2}{3}$, where n_2 is the number of vertices in G. The final guard set is $\Gamma_1 \cup \Gamma_2$ and has size at most $\frac{n_1}{3} + \frac{n_2}{3} = \frac{n}{3}$. Since removing vertices cannot decrease the hop distance between two faces, all marked quadrilaterals in G are still 3-hop apart.

We now turn to the coloring method from Theorem 9 to find a guard set for G. However, we take greater care with quadrilateral faces in triangulating G and constructing the matchings M_{12} and M_{34}, to ensure that $M_{12} \cup M_{34}$ actually guards every face of G instead of just the non-quadrilateral faces. Together with Γ_{12} and Γ_{34}, this then gives us three guard sets of total size n_2, which means the smallest of the three has size at most $\frac{n_2}{3}$.

We construct a triangulation G' from G as in the proof of Theorem 9, with one exception. If a quadrilateral q does not share a boundary edge with a triangle, we merge it with one of its neighboring faces f by removing the edge (u, v) separating them (see Figure 6). The result is a face with at least 7 sides, since f was not a triangle and all quadrilaterals are further apart. Let $w_f \neq v$ be the other neighbor of u along the boundary of f, and $w_q \neq v$ the other neighbor of u along the boundary of q. We insert edges (v, w_f), (v, w_q), and (w_f, w_q), then triangulate the rest of the face as usual.

Next, we four-color G' and consider the resulting coloring of G. Note that the edges we removed could be monochromatic, but this is not a problem. Let G_{12} and G_{34} be the subgraphs of G induced by all vertices with colors in $\{c_1, c_2\}$ and $\{c_3, c_4\}$, respectively. First, suppose M_{12} is an arbitrary maximal matching in G_{12} and M_{34} in G_{34}. Since each face of G contained a triangle in G', it has vertices of at least three different colors. Therefore we still obtain guard sets Γ_{12} and Γ_{34} by taking the matchings and adding an edge incident to every vertex of the right colors not in the corresponding matching. Similarly, as argued in the proof of Theorem 9, $M_{12} \cup M_{34}$ guards all non-quadrilateral faces of G. We now show how to pick initial edges for M_{12} and M_{34} such that $M_{12} \cup M_{34}$ also guards the marked quadrilateral faces of G. Recall that the unmarked quadrilateral faces of G are already guarded by Γ_1.

Initially, M_{12} and M_{34} are empty. If a marked quadrilateral q shares a boundary edge with a triangle t, then the vertices of t have three distinct colors. Therefore one of the edges of t must belong to G_{12} or G_{34}, and we add this edge to the corresponding matching. If q does not share an edge with a triangle, we merged it with a neighboring face by removing edge (u, v). Suppose that u has a color in $\{c_1, c_2\}$. Since three of its neighbors in $G - v$, w_f, and w_q – formed a triangle in G', one of them must also have a color in $\{c_1, c_2\}$, and we add this edge to M_{12}. If u has a color in $\{c_3, c_4\}$, we add the corresponding edge to M_{34}.

Thus, we seed M_{12} and M_{34} with edges that together guard all marked quadrilateral faces of G. We then complete these sets to maximal matchings by greedily adding edges of G_{12} and G_{34}, respectively. This makes $M_{12} \cup M_{34}$ a third guard set. The only thing left to argue is that none of the seed edges share an endpoint. This is guaranteed by the 3-hop distance between marked quadrilaterals in G; since each seed edge is incident to a marked quadrilateral, two seed edges sharing an endpoint would give a 2-hop path between two marked quadrilateral faces. ◀

5 Conclusion

Our main contribution lies in the development of techniques that allowed us to improve the upper bound on the number of edge guards that suffice to guard a plane graph. The role of quadrilateral faces in the size of these guard sets is intriguing. Of our bounds, one depends on the number of quadrilateral faces, while the other does not. The first bound $(\frac{n}{3} + \frac{\alpha}{9})$ almost matches the lower bound for graphs with few quadrilateral faces, while the second bound $(\frac{3n}{8})$ is stronger for graphs with many quadrilaterals – the two bounds balance at $\alpha = \frac{3n}{8}$ since $\frac{n}{3} + \frac{3n}{72} = \frac{3n}{8}$. It is interesting that quadrilateral faces are the limiting factor in all techniques based on graph colorings. In contrast, our iterative technique appears to be limited by the local nature of the operation. Thus, the solution may lie in a more global approach that does not stumble over quadrilateral faces.

We leave as an open question to close the gap between the upper and lower bounds, both for maximal planar graphs and general planar graphs.

References

1. Kenneth Appel and Wolfgang Haken. *Every planar map is four colorable*, volume 98 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1989. With the collaboration of J. Koch. doi:10.1090/conm/098.
2. Oleg V. Borodin. Structure of neighborhoods of an edge in planar graphs and the simultaneous coloring of vertices, edges, and faces. *Matematicheskie Zametki*, 53(5):35–47, 1993.
3. Prosenjit Bose, David G. Kirkpatrick, and Zaiqing Li. Worst-case-optimal algorithms for guarding planar graphs and polyhedral surfaces. *Computational Geometry: Theory and Applications*, 26(3):209–219, 2003.
4. Prosenjit Bose, Thomas C. Shermer, Godfried T. Toussaint, and Binhai Zhu. Guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7:173–185, 1997.
5. Václav Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18:39–41, 1975.
6. Hazel Everett and Eduardo Rivera-Campo. Edge guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7:201–203, 1997.
7. Steve Fisk. A short proof of Chvatal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.
8. Henri Lebesgue. Quelques conséquences simple de la formula d'Euler. *Journal de Mathématiques Pures et Appliquées*, 19:27–43, 1940.
9. Joseph O'Rourke. Galleries need fewer mobile guards: a variation on chvatal's theorem. *Geometriae Dedicata*, 14:273–283, 1983.
10. Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.
11. Jorg Sack and Jorge Urrutia, editors. *Handbook of Computational Geometry*. North-Holland, 2000.
12. Thomas C. Shermer. Recent results in art galleries. *Proceedings of IEEE*, 80:1384–1399, 1992.
13. Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 2017.

Sparse Weight Tolerant Subgraph for Single Source Shortest Path

Diptarka Chakraborty
Computer Science Institute of Charles University, Prague, Czech Republic
diptarka@iuuk.mff.cuni.cz

Debarati Das
Computer Science Institute of Charles University, Prague, Czech Republic
debaratix710@gmail.com

Abstract

In this paper we address the problem of computing a sparse subgraph of any weighted directed graph such that the exact distances from a designated source vertex to all other vertices are preserved under bounded weight increment. Finding a small sized subgraph that preserves distances between any pair of vertices is a well studied problem. Since in the real world any network is prone to failures, it is natural to study the fault tolerant version of the above problem. Unfortunately, it turns out that there may not always exist such a sparse subgraph even under single edge failure [Demetrescu *et al.* '08]. However in real applications it is not always the case that a link (edge) in a network becomes completely faulty. Instead, it can happen that some links become more congested which can be captured by increasing weight on the corresponding edges. Thus it makes sense to try to construct a sparse distance preserving subgraph under the above weight increment model where total increase in weight in the whole network (graph) is bounded by some parameter k. To the best of our knowledge this problem has not been studied so far.

In this paper we show that given any weighted directed graph with n vertices and a source vertex, one can construct a subgraph of size at most $e\cdot(k-1)!2^k n$ such that it preserves distances between the source and all other vertices as long as the total weight increment is bounded by k and we are allowed to only have integer valued (can be negative) weight on edges and also weight of an edge can only be increased by some positive integer. Next we show a lower bound of $c\cdot 2^k n$, for some constant $c \geq 5/4$, on the size of such a subgraph. We further argue that the restrictions of integral weight and integral weight increment are actually essential by showing that if we remove any one of these two we may need to store $\Omega(n^2)$ edges to preserve the distances.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Shortest path, fault tolerant, distance preserver, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.15

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787.

Acknowledgements The first author would like to thank Pavan Aduri and Vinodchandran N. Variyam for some helpful discussions during initial phase of this work and a special thank to Pavan Aduri for suggesting to study this problem. Authors also thank Keerti Choudhary, Shahbaz Khan and Michal Koucký for many valuable suggestions and comments.

1 Introduction

In the real world, networks are prone to failures and most of the time such failures are unpredictable as well as unavoidable in any physical system such as communication network or road network. For this reason, in the recent past, researchers study many graph theoretic questions like connectivity [29, 27, 4, 23], finding shortest distance [18], building data structure that preserves approximate distances [25, 16, 14, 19, 7, 10, 3] etc. under the fault tolerant model. Normally such failures are much smaller in number comparative to the size of the graph. Thus we can associate a parameter to capture the number of edge or vertex failures and try to build fault tolerant data structures of size depending on this failure parameter for various graph theoretic problems.

Unfortunately, in case of single source shortest path problem, it is already known from [18] that there are graphs with n vertices for which to preserve the distances under even single edge failure, we need to store a subgraph of size at least $\Omega(n^2)$. On the other hand, in case of reachability problem we know a construction of connectivity preserving subgraph of size only $O(2^k n)$ [4] where k is the number of edge failures. However, in the real world it is not always the case that there are failures of edges or vertices. Instead, for weighted graphs, weight of any edge or vertex can be increased. For simplicity, we consider weight to be only on the edges. In general, weight of an edge captures aspect like congestion on a particular link in a network. So it is quite natural to consider the scenario when some links (edges) become more congested. Again the good thing is that most of the time such congestion is bounded, i.e., over a network total increase in congestion is bounded because of many reasons like bounded maximum number of consumers present in a network at any particular time. One can easily capture the increase in congestion by a parameter k that bounds the amount of increase in weight of edges over the whole graph. Occurrence of such bounded congestion motivates us to study the single source shortest path problem under this model.

In this paper we initiate the study of single source shortest path problem for weighted directed graphs in the bounded weight increment model. The main goal is to find a sparse subgraph that preserves the (shortest) distance between a designated source and any other vertex under weight increment. We formally define such a subgraph below.

▶ **Definition 1** (k-WTSS)**.** Given a graph G along with a weight function w, a source vertex $s \in V(G)$ and an integer $k \geq 1$, a subgraph $H = (V(G), E')$ where $E' \subseteq E(G)$ is said to be a *k-Weight Tolerant single source Shortest-path Subgraph (k-WTSS)* of G if for any weight increment function $I : E(G) \to \mathbb{N}$ such that $\sum_{e \in E(G)} I(e) \leq k$, the following holds: for the weight function defined by $w'(e) = w(e) + I(e)$ for all $e \in E(G)$,

$$dist_{G,w'}(s,t) = dist_{H,w'}(s,t) \text{ for any } t \in V(G).$$

Though in the above definition we restrict ourselves to an increment function whose range is $\mathbb{N}$, one can naturally extend the definition to any range of increment functions. However, if we take the range to be rational numbers then there may not exist any sparse k-WTSS even for $k = 1$ (see Section 6). This is the reason why we consider such restriction on increment function in the above definition.

Single source shortest path is one of the most fundamental problems in Graph Theory as well as in computer science. Thus construction of a sparse k-WTSS is interesting from both the theoretical and practical point of view. One can also view the problem of finding k-WTSS as a generalization of finding k-Fault Tolerant (single source) Reachability Subgraph (k-FTRS) for which optimal solution is known due to [4]. If in the given graph one assigns zero weight on the edges, then any k-WTSS of that graph will also be a k-FTRS. This

is because we can view each edge fault as incrementing weight by one and then it is easy to see that for any vertex t there exists an $s-t$ path iff the shortest distance between s and t is zero under this reduction. This fact also motivates the study of constructing k-WTSS because k-FTRS has several applications like fault tolerant strong-connectedness [4], dominators [24, 4], double dominators [34] etc. We have already mentioned that it is possible to represent congestion in any network by incrementing weight of the links. Unlike to the edge faults, which can be thought of as an independent and equally likely process, congestion is strongly time-variant (e.g. [33]). For example, if at any point of time some edge is congested and it has small out-degree then at the next time step some or all of its out edges will also be congested. However the good thing about k-WTSS subgraph is that no matter how the congestion occurs or propagates, as long as the total weight increment is bounded by k, a k-WTSS subgraph will preserve the distances. Hence for the practical purposes it is always useful to construct (and store) a sparse k-WTSS (if exists). In spite of being an appealing problem, to the best of our knowledge the problem of constructing sparse k-WTSS has not yet been studied. Although, a lot of research has been done on different versions of shortest path problem under *dynamic* weight changes [32, 30, 17, 12] which have found several applications in network optimization, internet routing, databases and many more.

The main contribution of this paper is to provide an efficient construction of a sparse k-WTSS for any $k \geq 1$, where sparsity of k-WTSS depends on the parameter k.

▶ **Theorem 2.** *There exists an $O((k)^k m^2 n)$-time algorithm that for any given integer $k \geq 1$, and a given directed graph G with n vertices and m edges along with a weight function $w : E(G) \to \mathbb{Z}$ and a source vertex s, constructs a k-WTSS of G with at most $O_k(n)$ edges where $O_k(\cdot)$ notation denotes involvement of a constant that depends only on the value of k. Moreover, in-degree of every vertex in this k-WTSS will be bounded by $e \cdot (k-1)!2^k$.*

Next, we prove a lower bound of $c \cdot 2^k n$ for some constant $c \geq 5/4$, on the size of k-WTSS.

▶ **Theorem 3.** *For any positive integer $k \geq 2$, there exists a positive integer n_k so that for all $n > n_k$, there exists a directed graph G with n vertices and a weight function $w : E(G) \to \mathbb{Z}$, such that its k-WTSS must contain $c \cdot 2^k n$ many edges for some constant $c \geq 5/4$.*

We provide the proof of Theorem 3 in the full version [13]. Note that as we have previously argued that the construction of k-WTSS implies a construction of k-FTRS, so $2^k n$ lower bound on the size of a k-FTRS due to [4] also directly gives the same lower bound on size of a k-WTSS. In the above theorem we slightly strengthen that lower bound by a constant factor for our problem. We also show that considering rational valued weight function or rational valued weight increment function (instead of integer valued as in the above two theorems) makes the problem of finding sparse k-WTSS impossible. More specifically, we show that in both the cases there are graphs with n vertices for which any k-WTSS must be of size at least $\Omega(n^2)$ even for $k = 1$ (see Section 6).

One can further relax our model by also allowing decrement operation on edge weight. Weight decrement is also natural in real life applications because for any network it is possible that some links become less congested. Unfortunately, one can easily show that there are graphs for which there is no sub-quadratic sized subgraph that preserves the distances from a single source under this relaxed model. Readers may refer to the full version [13] for the details.

Related works

Single source shortest path is a well studied problem under the edge or vertex failure model. Similar to our definition of k-WTSS, one can easily define k-Fault Tolerant Shortest-path

Subgraph (k-FTSS) that preserves the distance information from a specific source vertex under at most k edge failures. Unfortunately, we know that there are weighted graphs for which no sparse k-FTSS exists even for $k = 1$, i.e., there are weighted graphs with n vertices for which any 1-FTSS must contain $\Omega(n^2)$ many edges [18]. This lower bound on size of 1-FTSS is even true for undirected graphs. However, better bounds are known for unweighted graphs for $k \leq 2$. Parter and Peleg [28] provided a construction of $O(n^{3/2})$ sized 1-FTSS and showed that this bound is optimal. Later, Parter [27] extended the construction to the case $k = 2$ for undirected graphs on the cost of weakening the bound and gave an algorithm to compute 2-FTSS of size $O(n^{5/3})$ along with a matching lower bound. This result has very recently been extended to directed graphs [23]. For general k, a construction of sub-quadratic sized FTSS is known for undirected unweighted graphs due to [10].

However, the situation is much better for single source reachability problem which is closely related to single source shortest path problem. Baswana *et al.* [4] showed that we can compute k-FTRS, which is a subgraph that preserves the reachability information from a given source under at most k edge failures, containing $2^k n$ many edges. They also provided a matching lower bound. We have already argued that computing k-FTRS can be reduced to computing k-WTSS and thus it is natural to ask whether a similar result also holds for k-WTSS. Another interesting related problem is to compute fault tolerant reachability oracle. It is trivial to see that using $O(2^k n)$ size k-FTRS [4] one can answer any reachability query in $O(n)$ time for any constant value of k. However for $k \leq 2$, $O(n)$ size data structure is known that can answer any single source reachability query in $O(1)$ time [24, 15].

Coming back to the shortest path problem, instead of preserving the exact distances (between any pair of vertices), if we consider to preserve the distances only approximately, then much better results are known. Such approximate distance preserving subgraphs are called *spanners*. Construction of spanners with both additive and multiplicative stretch have been studied extensively [5, 37, 1, 2]. Fault tolerant version of spanners were first introduced in the geometric setting [25]. For k edge failures, construction of a $(2l-1)$ multiplicative spanner of size $\widetilde{O}(kn^{1+1/l})$, for any $k, l \geq 1$, was provided in [14] whereas for k vertex failures, upper bound on size is known to be $\widetilde{O}(k^{2-1/l}n^{1+1/l})$ [19]. In case of single edge failure, for single source undirected graphs, construction of a $2n$ sized subgraph that preserves distances within a multiplicative factor of 3 is known [8]. Braunschvig *et al.* [11] initiated the study of additive spanners. For β-additive spanner, Parter and Peleg [29] provided a $\Omega(n^{1+\epsilon(\beta)})$ size lower bound where $\epsilon(\beta) \in (0, 1)$. For single source undirected graphs they also constructed a 4-additive spanner of size $O(n^{4/3})$ that is resilient to single edge failure. For single vertex failure, constructions of additive spanners were given in [26, 7]. Very recently, for any fixed $k \geq 1$, construction of a sub-quadratic size 2-additive spanner resilient to k edge or vertex failures has been shown for unweighted undirected graphs [10]. In the same paper, authors also show that to achieve $O(n^{2-\epsilon})$ upper bound, one must allow $\Omega(\epsilon k)$ additive error.

Designing distance oracle is another important problem and also has been studied in edge failure model. The objective is to build a fault tolerant data structure that can answer queries about the distances in a given graph. For single edge failure the problem was first studied in [18]. Construction of $\widetilde{O}(n^2)$-space and $O(1)$-query time oracle is known for single edge failure due to [6]. In case of dual edge failures, near optimal $\widetilde{O}(n^2)$ size and $\widetilde{O}(1)$-query time oracle was given in [20]. The problem has also been studied under the restriction of bounded edge weights [22, 35]. For general k edge failures, Bilò *et al.* [9] gave a construction of $O(kn \log^2 n)$ size data structure that can report distance from a single source within multiplicative factor of $(2k+1)$ in time $O(k^2 \log^2 n)$.

Another closely related problem is the replacement path problem where given a source and destination vertex and an edge, the objective is to find a path from source to destination avoiding that particular given edge. Though the problem was initially defined for single edge failure, later it was extended to multiple edge failures also. Readers may refer to [36, 31, 22, 35] for recent progresses on this problem.

Our technique

Before exhibiting the technique behind our result, we first state a simple observation. If we just store any shortest path tree rooted at s, then even after k weight increment that tree will preserve the distance from s to t (for any t) within k additive error. It is also necessary to include a shortest path tree inside a k-WTSS, otherwise we can never hope to get exact distances even when $k = 0$. Now since weight of any path can be increased by at most k, after including any shortest $s - t$ path in a k-WTSS it is not required to include another $s - t$ path that has weight more than or equal to $dist(s, t) + k$.

We argue that for the construction of k-WTSS it suffices to concentrate on any single vertex t and try to build a subgraph such that the distance between s and t is preserved under weight increment and we call such a subgraph a k-WTSS(t). This is because of the application of Locality Lemma (see Section 4), a variant of which also appears in [28, 29, 4]. Locality Lemma actually says slightly more, that if we can construct such a subgraph for any vertex t with an additional property that in-degree of t in the subgraph is bounded by some value c, then we can get a k-WTSS of size at most cn.

So from now on we can only talk about constructing k-WTSS(t). Let us take a toy example which provides a motivation behind our technique. Let the input graph be G and $dist(s, t) = d$. Suppose G is such that it can be decomposed into $k + 1$ disjoint subgraphs $G_0, \cdots, G_k$ where for $0 \leq i \leq k - 1$, G_i contains all the $s - t$ paths of weight $d + i$ present in G and any $s - t$ path in G_i has weight exactly $d + i$. In general such a decomposition may not exist. However, if it exists then it is not hard to get such a decomposition. Now given such a decomposition, we compute a k-FTRS(t) of G_0 and for $i \in [k - 1]$, a $(k - i - 1)$-FTRS(t) of G_i and then take the union of them. We claim that the obtained subgraph will be a k-WTSS(t). Say after weight increment, the (shortest) distance between s and t is $d + j$ for $1 \leq j < k$. Our assumption on j is justified because $j = 0, k$ cases are trivial as we have included k-FTRS(t). For a similar reason we can also assume that all the original shortest paths now have weight at least $d + j + 1$. Without loss of generality we further assume that no weight increment happens on the edges of the current shortest path. The justification of this assumption is provided in the full version [13]. Due to our assumption on decomposition of G, we know that the total increase in weight on the edges of G_j is bounded by $k - (j + 1)$ which also implies that at most $k - (j + 1)$ many edges of G_j are affected by weight increment. This is because our increment function is integer valued. Note that this is the place where integer valued increment plays a crucial role. However by our construction, we have included $(k - j - 1)$-FTRS(t) of G_j in our subgraph. Thus even if we remove those affected edges, since there is a path in G_j on which there is no weight increment, by the definition of $(k - j - 1)$-FTRS(t) there will be a surviving path included in our constructed subgraph. This proves the correctness. Also by the result of [4], in-degree of t of each $(k - i - 1)$-FTRS(t) of G_i is bounded by 2^{k-i-1} and hence total in-degree of t in the constructed k-WTSS(t) is bounded by 2^{k+1}. Hence we get a k-WTSS of size at most $2^{k+1}n$.

We have already mentioned that there may not exist the above type of decomposition for an arbitrary graph. In general, if we consider a subgraph by taking all the $s - t$ paths upto some specific weight, then that particular subgraph may also contain a $s - t$ path with larger

weight. At this point the argument stated in the last paragraph fails completely. However, the nice thing is that if we just consider all the shortest paths and build a subgraph then it is true that there will not be any $s-t$ path of larger weight in that subgraph. Now if we use the construction of k-FTRS on this shortest path subgraph, then we can guarantee the preservation of distances as long as the distances do not change even after the weight increment. Though if the distance changes, we cannot say anything. This is the main challenge that we overcome in our algorithm. For that purpose we use the properties of the farthest min-cut of the shortest path subgraph.

Baswana *et al.* [4] used the concept of farthest min-cut to construct k-FTRS. In their work, they first computed a series of k farthest min-cuts by taking source sets in some nested fashion. Then they calculated a max-flow from the final source set and kept the incoming edges of t having non-zero flow. We further exploit their technique in this paper to get our algorithm. We consider the shortest path subgraph and then compute a series of farthest min-cuts similar to [4]. However as mentioned in the last paragraph, in this way we just get k-FTRS(t) of the shortest path subgraph. Now let us take the farthest min-cut considering s as source. Since it is a (s,t)-cut of the shortest path subgraph, removal of it destroys all the shortest $s-t$ paths present in the original graph. Now if we again compute the shortest path subgraph, we will get a subgraph containing only $s-t$ paths of weight $d+i$, for some $i>0$. Then we can process this new subgraph as before to compute a sequence of k farthest min-cuts and remove the first one. We proceed in this way until we reach at a point that we are left with $s-t$ paths of weight at least $d+k$.

Now let us compare the situation with our previously described toy example. Removal of cut edges only helps us to generate some subgraph of each of G_i's. However computing k-FTRS(t) of just some subgraph of G_i's may not be sufficient to get k-WTSS(t). Thus for each G_i, we try to get a lot of subgraphs of it so that when we combine k-FTRS(t) of all of them, we get the same advantage that we got from computing $(k-i-1)$-FTRS(t) of G_i in the toy example. One way of getting a lot of subgraphs of G_i is to try out removal of different cuts (not just the farthest one). Obviously we cannot try for all possible cuts, because there can be too many. Moreover, each time to reach at a subgraph of weight $d+i$ we may have to remove a series of $i-1$ cuts. As a result we may end up with exponentially many choices on removal of cuts to get all possible subgraphs of G_i.

The good thing is that it suffices to use just a series of k farthest min-cuts computed before for the purpose of removal. A stronger claim is formally stated in Lemma 15. This will reduce the number of choices to only k^i for any fixed G_i. In our algorithm we establish a slightly better bound on the number of subgraphs of G_i needed to be considered to construct a k-WTSS(t). In the proof we use k-tuples to efficiently enumerate all of these subgraphs. Informally, a subgraph indexed by a specific k-tuple consists of only the $s-t$ paths survived after removal of a set of cut sets identified by the value of the coordinates of the k-tuple. Now after getting those subgraphs we apply a construction similar to that of k-FTRS(t) from [4] to get a bound on in-degree of t. We emphasize that actually we cannot directly apply algorithm of [4] in a black box fashion on each of the subgraphs of G_i that we consider, because in that case it will not give us the claimed bound.

In this paper we consider k-WTSS with respect to the weight increment on edges. Instead, it is also possible to take weights on the vertices and perform increment over them. However, one can directly apply our result by splitting each vertex v into two vertices v_i and v_o where all the incoming and outgoing edges of v are respectively directed into v_i and directed out of v_o, and then considering an edge (v_i, v_o) with the weight equal to that on v.

Organization of the paper

We discuss useful notations and some already known results about farthest min-cut in Section 2. Then in Section 3 we provide an algorithm to compute farthest min-cut of the shortest path subgraph and a few important properties about it. Next in Section 4, we reduce the problem of finding k-WTSS to that of finding k-WTSS(t) for some specific vertex t using Locality Lemma. Finally we prove Theorem 2 in Section 5. We also present several lower bound results in Section 6.

2 Preliminaries

Notations:

For any positive integer r, we denote the set $\{1, 2, \cdots, r\}$ by $[r]$. Throughout this paper we use $\mathbb{N}$ to indicate the set of natural numbers including zero. For any k-tuple σ and $i \in [k]$, we use the notation $\sigma(i)$ to denote the value of the i-th coordinate of σ. Given a directed graph $G = (V, E)$ on a set of vertices of size $|V| = n$ and a set of edges of size $|E| = m$ with a weight function w defined on the set of edges, a source vertex $s \in V$ and a destination vertex $t \in V$, we use the following notations throughout this paper.

- $V(G), E(G)$: the set of vertices and edges of G respectively.
- $w(P)$: weight of any path P.
- $dist_{G,w}(x, y)$: the shortest distance between any two vertices x and y in G when weight of each edge is defined by the weight function w.
- $G + (u, v)$: the graph obtained by adding an edge (u, v) to the graph G.
- $G \setminus F$: the graph obtained by removing the set of edges F from the graph G.
- $Out(A)$: the set of all vertices in $V \setminus A$ having an incoming edge from $A \subseteq V$.
- $In\text{-}Edge(A)$: the set of edges incoming to $A \subseteq V$.
- $P[x, y]$: the subpath of a path P from a vertex x to y.
- $P \circ Q$: the path formed by concatenating paths P and Q assuming the fact that last vertex of P is same as first vertex of Q.
- $E(f)$: support of the flow f, i.e., the set of edges e such that $f(e) \neq 0$.
- $MaxFlow(G, S, t)$: any maximum valued flow in G from a source set S to t.
- G_{short} : the shortest path subgraph of G, i.e., union of all shortest $s - t$ paths in G.
- $ShortMaxFlow(G, S, t)$: any maximum valued flow returned by $MaxFlow(G_{short}, S, t)$.

The following definition introduces the notion of k-WTSS with respect to a fixed vertex t.

▶ **Definition 4** (k-WTSS(t))**.** Given a graph G with a weight function w, a source vertex $s \in V(G)$, another vertex $t \in V(G)$ and an integer $k \geq 1$, a subgraph $H_t = (V(G), E')$ where $E' \subseteq E(G)$ is said to be a *k-WTSS(t)* of G if for any weight increment function $I : E(G) \to \mathbb{N}$ such that $\sum_{e \in E(G)} I(e) \leq k$, the following holds: for the weight function defined by $w'(e) = w(e) + I(e)$ for all $e \in E(G)$, $dist_{G,w'}(s, t) = dist_{H_t,w'}(s, t)$.

The restriction on the range of the increment function to $\mathbb{N}$ is justified because only in that case we can hope for a sparse k-WTSS(t) (see Section 6). However one can easily extend the above definition for increment function $I : E(G) \to \mathbb{R}$. Following is an alternative definition of k-WTSS in terms of k-WTSS(t).

▶ **Definition 5.** A subgraph H is a *k-WTSS* of G iff it is a k-WTSS(t) for all $t \in V(G)$.

2.1 Max-flow and farthest min-cut

The algorithm described in this paper heavily exploits the connection between min-cut, max-flow and the number of edge disjoint paths present in a graph. Let us start with the following well known fact of Graph Theory.

▶ **Theorem 6.** *In any graph with unit capacity on edges, there is a flow of value r from a source set S to a destination vertex t if and only if there exist r edge disjoint paths that originate from the set S and terminate at t.*

For the sake of clarity, we emphasize that though we talk about weighted graphs, throughout this paper we will use capacity functions on edges that take values only from $\{0, 1\}$.

▶ **Definition 7** ((S,t)-min-cut)**.** In any graph G an (S,t)-*cut* is a set of edges $C \subseteq E(G)$ such that every path from any vertex $s \in S$ to t must pass through some edge in C. An (S,t)-cut is called (S,t)-*min-cut* if it has the smallest size among all other (S,t)-cuts.

Any (S,t)-cut C partitions the vertex set $V(G)$ into two subsets $A(C)$ and $B(C)$ where $A(C)$ is the set of all the vertices reachable from S in $G \setminus C$ and $B(C) = V(G) \setminus A(C)$. Note that $S \subseteq A(C)$ and $t \in B(C)$. From now on, we assume this pair of vertex sets $(A(C), B(C))$ to be output of a function $Partition(G, C)$. For our purpose we do not just consider any (S,t)-min-cut, instead we consider the farthest one.

▶ **Definition 8** (Farthest Min Cut [21])**.** Let S be a source set and t be a destination vertex in any graph G and suppose for any (S,t)-min-cut C, $(A(C), B(C)) = Partition(G, C)$. Any (S,t)-min-cut C_{far} is called *farthest min-cut*, denoted by $FMC(G, S, t)$, if for any other (S,t)-min-cut C, it holds that $A(C) \subsetneq A(C_{far})$.

The following lemma given by Ford and Fulkerson [21] establishes the uniqueness of farthest min-cut and also provides an algorithm to compute it.

▶ **Lemma 9** ([21])**.** *Suppose f be a max-flow in G from any source set S to t and G_f be the corresponding residual graph. If B is the set of vertices from which there is a path to t in G_f and $A = V(G) \setminus B$, then the set C of edges that start at A and terminate at B is the unique farthest (S,t)-min-cut.*

3 Farthest Min-cut of Shortest Path Subgraph

3.1 Computing farthest min-cut of shortest path subgraph

In this section we give an algorithm to find the farthest min-cut of the shortest path subgraph of a given graph. We are given a weighted directed graph G with two vertices s and t. The weight of each edge of G is defined by a weight function $w : E(G) \to \mathbb{R}$. Let $dist_{G,w}(s,t) = d$. We denote the set of all $s-t$ paths of weight d under the weight function w by $\mathcal{P}_d$ and the corresponding underlying subgraph (just the union of all the paths in $\mathcal{P}_d$) of G by G_{short}. More specifically, $V(G_{short}) = V(G)$ and $E(G_{short}) = \{e \mid e \in P \text{ for some } P \in \mathcal{P}_d\}$. For any graph G and two vertices s and t, for any source set $S \subseteq V(G_{short})$, *farthest min-cut of shortest path subgraph*, denoted as $FSMC(G, s, S, t)$ is defined by $FMC(G_{short}, S, t)$.

For any (S,t)-cut C of G_{short} we define the partition function by $ShortPartition(G, C) = Partition(G_{short}, C)$. It is easy to design (see the full version [13]) a simple $O(mn)$ time procedure which given G, s and t, generates the subgraph G_{short}. Then we can simply apply well known Ford-Fulkerson algorithm [21] on the subgraph G_{short} to find the $ShortMaxFlow(G, S, t)$ and $FSMC(G, s, S, t)$. The correctness of $FSMC(G, s, S, t)$ follows from applying Lemma 9 on the subgraph G_{short}.

3.2 Disjoint shortest path lemma

Let us choose any $r \in \mathbb{N}$ and then consider the following: Set $S_1 = \{s\}$ and for $i \in [r]$, define $C_i = FSMC(G, s, S_i, t)$, $(A_i, B_i) = ShortPartition(G, C_i)$ and $S_{i+1} = (A_i \cup Out(A_i)) \setminus \{t\}$. Let $E' \subseteq E(G)$ such that $E' = \{(u_1, v_1), \cdots, (u_r, v_r)\}$, where $(u_i, v_i) \in C_i$.

Now let us introduce an auxiliary graph $G' = G + (s, v_1) + \cdots + (s, v_r)$ and set $w(s, v_i) = dist_{G,w}(s, v_i)$ for $i \in [r]$. Suppose f is a max-flow from S_{r+1} to t in the shortest path subgraph of G and $\mathcal{E}(t)$ be the set of incoming edges of t having nonzero flow value assigned by f. Now consider a new graph $G^* = (G' \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$.

▶ **Lemma 10.** *There will be at least $r + 1$ disjoint paths in G^* each of weight equal to $dist_{G,w}(s, t)$.*

Note that a similar claim was shown in [4]. However, our claim is slightly more general because we consider an edge set E' where the edges belong to E' may not lie on a single $s - t$ path in G and also we comment on the weight of the disjoint paths. Both of these requirements are crucial for the proof in Section 5. Fortunately, the proof in [4] does not rely on the fact that the edges (u_i, v_i)'s are part of a single $s - t$ path.

4 Construction of k-WTSS and Locality Lemma

Let us first recall the problem. We are given a graph G along with a weight function $w : E(G) \to \mathbb{Z}$ and a source vertex s. Now suppose for every $e \in E(G)$, $w(e)$ is increased by some arbitrary weight increment function $I : E(G) \to \mathbb{N}$ such that total increase in weight is bounded by k, i.e., $\sum_{e \in E(G)} I(e) \leq k$ and we denote the new weight function (after increase in weight) by w' where $w'(e) = w(e) + I(e)$. The problem is to find a subgraph H such that for any vertex $t \in V(G)$ in H there always exists an $s - t$ path of weight $dist_{G,w'}(s, t)$. We call this subgraph H a k-WTSS. Now if we just want the requirement of existence of a path in H to be true for a fixed vertex t instead of all vertices, then we call such a subgraph k-WTSS(t). In this section we reduce the problem of finding k-WTSS to the problem of finding k-WTSS(t) for any fixed vertex $t \in V(G)$. The following lemma, a variant of which also appears in [4], serves our purpose.

▶ **Lemma 11** (Locality Lemma)**.** *Let there be an algorithm $\mathcal{A}$ that given a graph G and a vertex $t \in V(G)$, generates a subgraph H_t of G such that:*

- *H_t is a k-WTSS(t); and*
- *in-degree of t in H_t is bounded by a constant c_k.*

Then one can generate a k-WTSS of G such that it has only $c_k \cdot n$ edges.

5 Construction of k-WTSS(t)

In this section we provide an algorithm to compute a k-WTSS(t) for any fixed vertex $t \in V(G)$ where source vertex is s. Without loss of generality let us first assume the following.

▶ **Assumption 12.** *The out degree of source vertex s is 1 and the out degree of all other vertices is bounded by 2.*

For any graph G if $|Out(s)| > 1$ then to satisfy our previous assumption, we can simply add a new vertex s_0 and add an edge (s_0, s) and set $w(s_0, s) = 0$. Then make this new vertex s_0 as our new source. For the justification on the bound on out degree of other vertices, we refer the readers to the full version [13]. Note that the reduction process blows up the number of

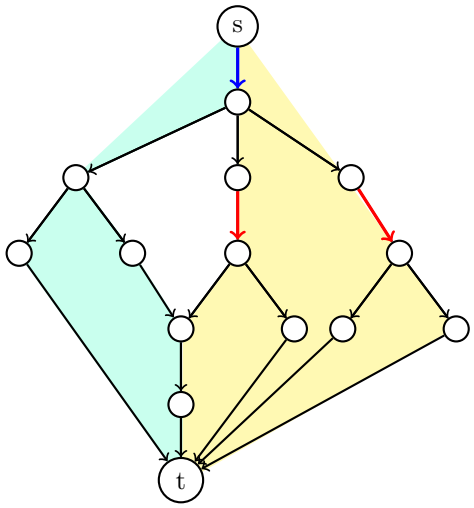


Figure 1 Region shaded with green color represents G_σ for $\sigma = (1, -1, \cdots, -1)$ whereas yellow colored region is the shortest path subgraph of G. The edges of $C_{(-1,\cdots,-1),1}$ and $C_{(-1,\cdots,-1),2}$ are colored with blue and red respectively. G_σ is obtained by removing red colored edges.

vertices from n to $O(m)$. However, since we bound the in-degree of t in k-WTSS(t) for any $t \in V(G)$, by a value independent of the number of vertices in the graph, our result remains unaffected.

5.1 Description of the algorithm

Before describing the algorithm let us introduce some notations that we will use later heavily. Consider any k-tuple $\sigma \in \{-1, 0, 1, \cdots, k\}^k$ such that if $\sigma(i) = -1$ (where $\sigma(i)$ denotes the i-th coordinate of σ) then for all $i' > i$, $\sigma(i') = -1$ and if $\sigma(i) \neq -1$ then for all $i' < i$, $\sigma(i') \neq -1$. We use these k-tuples to efficiently enumerate all the subgraphs of G for which we want to calculate farthest min-cuts. Informally, a subgraph indexed by σ consists of only the $s-t$ paths survived after removal of a set of cut sets identified by $\sigma(i)$'s. For any such $\sigma \in \{-1, 0, 1, \cdots, k\}^k$ and $r \in [k]$, we recursively define the subgraph G_σ, set of source vertices $S_{\sigma,r}$ and edge set $C_{\sigma,r}$ as follows: if $\sigma = (-1, -1, \cdots, -1)$, G_σ is the union of all $s-t$ paths in G, starting with $S_{\sigma,1} = \{s\}$, for any $r \in [k]$ define $C_{\sigma,r} = FSMC(G_\sigma, s, S_{\sigma,r}, t)$, $S_{\sigma,r+1} = (A \cup Out(A)) \setminus \{t\}$ where $(A, B) = ShortPartition(G_\sigma, C_{\sigma,r})$. For $\sigma \neq (-1, \cdots, -1)$, G_σ is the union of all $s-t$ paths in $G_{\sigma'} \setminus C_{\sigma',\sigma(i)+1}$, where $i = \min\{i' \mid \sigma(i') = -1\}$ and

$$\sigma'(i') = \begin{cases} \sigma(i') & \text{if } i' < i-1 \\ -1 & \text{otherwise} \end{cases}$$

Now starting with $S_{\sigma,1} = \{s\}$, if there exists a $s-t$ path of weight $d + i - 1$ then for any $r \in [k]$ define $C_{\sigma,r} = FSMC(G_\sigma, s, S_{\sigma,r}, t)$, $S_{\sigma,r+1} = (A \cup Out(A)) \setminus \{t\}$ where $(A, B) = ShortPartition(G_\sigma, C_{\sigma,r})$; else set $C_{\sigma,r} = \phi$. We refer the reader to Figure 1 for the better understanding about the graph G_σ.

We are given a weighted directed graph G with a weight function w and a source vertex s and a destination vertex t. The weight of each edge of G is defined by the weight function $w : E(G) \to \mathbb{Z}$. Overall our algorithm performs the following tasks: For different values of $\sigma \in \{-1, 0, \cdots, k\}^k$ it computes the sets $C_{\sigma,i}$ and $S_{\sigma,i}$ for $i \in [k]$. Then for each such σ, it

computes max-flow in the shortest path subgraph of G_σ by considering $S_{\sigma,k}$ as source and add the edges incident on t with non-zero flow to a set $\mathcal{E}(t)$. At the end, our algorithm returns the subgraph $H_t = (G \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$.

Our algorithm performs the above tasks in the recursive fashion. Starting with $\sigma = (-1, \cdots, -1)$, it first considers the shortest path subgraph of $G_\sigma = G$ and performs k iterations on it. At each iteration it computes the farthest min-cut $C_{\sigma,i}$ by considering $S_{\sigma,i}$ as source and t as sink starting with $S_{\sigma,1} = \{s\}$. Then it updates the graph by removing the edges present in $C_{\sigma,i}$ and passes this new graph in the next recursive call. Before the recursive call it also updates the σ by incrementing the value of $\sigma(j)$ by one and passes the updated value of σ to the recursive call. Here j is a parameter which denotes that the smallest coordinate of σ that has value -1. Initially j was set to 1 and before the next recursive call we increment its value by one. At the end of each iteration our algorithm updates the source set to $S_{\sigma,i+1}$ by including end points of all the edges present in the cut $C_{\sigma,i}$ in the set $S_{\sigma,i}$. At the end of k iterations, the algorithm computes max-flow in the shortest path subgraph of G_σ by considering $S_{\sigma,k}$ as source and add the edges incident on t with non-zero flow to a set $\mathcal{E}(t)$.

5.2 Correctness proof

Let us start with the following simple observation.

▶ **Observation 13.** *For any $\sigma \in \{-1, 0, \cdots, k\}^k$, any $s-t$ path in G_σ must have weight at least $d + i - 1$ where $i = \min\{i' \mid \sigma(i') = -1\}$.*

Note that the above observation is true only because we consider the range of our weight function w to be $\mathbb{Z}$. Otherwise above observation will trivially be false.

Now let us consider any increment function $I : E(G) \to \mathbb{N}$ such that $\sum_{e \in E(G)} I(e) \leq k$ and then denote the set of edges with non-zero value of the function I by F, i.e., $F = \{e \in E(G) | I(e) > 0\}$. So clearly $|F| \leq k$. Now suppose $dist_{G,w'}(s,t) = d' = d + j$ for some $0 \leq j \leq k$ where $w'(e) = w(e) + I(e)$. Thus we need to show that there also exists an $s-t$ path of weight d' in the subgraph H_t under the new weight function w'.

Suppose P be an $s-t$ path in G such that $w'(P) = d' = d + j$. For simplicity let us make the following assumption.

▶ **Assumption 14.** *For all $e \in P$, $I(e) = 0$, i.e., $w'(P) = w(P)$.*

▶ **Lemma 15.** *One of the following three cases must be satisfied.*

1. *There exists a σ such that P belongs to the subgraph G_σ where $\sigma(j) = -1$ and for some $r \in [k]$, the last edge of P belongs to the edge set $C_{\sigma,r}$.*
2. *There exists a σ such that P belongs to the subgraph G_σ where $\sigma(j+1) = -1$, $\sigma(j) \neq -1$ and there is no $i \in [j-1]$ such that $\sigma(i) \geq k - j + i - 1$.*
3. *There exists a σ such that P belongs to the subgraph G_σ where if $i = \min\{i' \mid \sigma(i') = -1\}$ then $i \leq j$ and for all $i' < i$, $\sigma(i') < k - j + i' - 1$ and P passes through all the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k-j+i-1}$.*

Now let us call the path P is of type-1, type-2 and type-3 respectively depending on which of the above three cases it satisfies.

Type-1. This case is the simplest among the three.

▶ **Lemma 16.** *If P is a type-1 path then P is contained in the subgraph H_t.*

Proof. Suppose (v,t) is the last edge of the path P. Now since $(v,t) \in C_{\sigma,r}$ for some σ and r, $(v,t) \in \mathcal{E}(t)$. Thus by the construction of the subgraph H_t, the edge (v,t) belongs to H_t. Also by the construction of the subgraph H_t, for all the vertices $u \neq t$, $In\text{-}Edge(u)$ belongs to H_t. Hence P must lie completely inside H_t. ◀

Type-2. By Observation 13 any path that belongs to the subgraph G_σ where $\sigma(j) \neq -1$ and $\sigma(j+1) = -1$, must have weight atleast $d+j$ under the weight function w. Now, since by Assumption 14 $w(P) = d+j$, P must pass through an edge $(u_r, v_r) \in C_{\sigma,r}$ for all $r \in [k]$. Consider an auxiliary graph $G'_\sigma = G_\sigma + (s,v_1) + \cdots + (s,v_k)$ and extend the weight function w as $w(s,v_r) = w(P[s,v_r])$. Then define another graph $G^*_\sigma = (G'_\sigma \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$. By Lemma 10 we claim the following.

▶ **Corollary 17.** *There will be $k+1$ edge disjoint paths in G^*_σ each of weight $w(P)$ under weight function w.*

Now we use the above corollary to conclude the following.

▶ **Lemma 18.** *If P is a type-2 path then there exists an $s-t$ path of weight d' in the subgraph H_t under the new weight function w'.*

Proof. By Corollary 17, we get $k+1$ edge disjoint paths $P_1, \cdots, P_{k+1}$ each of weight $w(P) = d+j$. Since $|F| \leq k$ where $F = \{e \in E(G) | I(e) > 0\}$, at least one of the $k+1$ edge disjoint paths, say P_1, must survive in $G^*_\sigma \setminus F$. If P_1 also belongs to the subgraph H_t then we are done. Otherwise P_1 must take some of the (s,v_r)'s as the first edge and the remaining portion $P_1[v_r,t]$ lies inside H_t. Now consider the following path $R = P[s,v_r] \circ P_1[v_r,t]$. By the construction of G'_σ, $w'(R) = w'(P_1) = w(P)$ and this completes the proof. ◀

Type-3. Suppose P is a type-3 path and thus belongs to G_σ for some σ where if $i = \min\{i' \mid \sigma(i') = -1\}$ then $i \leq j$ and for any $i' \leq i$, $\sigma(i') < k-j+i'-1$ and P passes through all the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k-j+i-1}$. P passes through an edge $(u_r, v_r) \in C_{\sigma,r}$ for all $r \in [k-j+i-1]$. For the ease of representation let us define $v_0 = s$. Now if there exists a positive integer $r \in [k-j+i-1]$ such that $w(P[v_{r-1}, u_r]) > dist_{G_\sigma,w}(v_{r-1}, u_r)$, replace the portions of path $P[v_{r-1}, u_r]$ by the $v_{r-1} - u_r$ path of weight $dist_{G_\sigma,w}(v_{r-1}, u_r)$. We do this until there is no such r and after that we call this new path as P'.

Now consider an auxiliary graph $G'_\sigma = G_\sigma + (s,v_1) + \cdots + (s, v_{k-j+i-1})$ and extend the weight function w as $w(s,v_r) = w(P[s,v_r])$. Next define another graph $G^*_\sigma = (G'_\sigma \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$. Now we use a similar but slightly more intricate argument than that used in case of Type-2 paths to conclude the following.

▶ **Lemma 19.** *If P is a type-3 path then there exists an $s-t$ path of weight $w(P) = d'$ in the subgraph H_t under the new weight function w'.*

5.3 Bound on the size of $\mathcal{E}(t)$

Before establishing the upper bound on the size of the set of edges $\mathcal{E}(t)$, let us define $C_{\sigma,k+1} = FSMC(G_\sigma, s, S_{\sigma,k+1}, t)$ for any $\sigma \in \{-1, 0, \cdots, k\}^k$.

Now since $FSMC(G_\sigma, s, S_{\sigma,i+1}, t) = FMC(G^{short}_\sigma, S_{\sigma,i+1}, t)$ for any $i \in [k]$ where G^{short}_σ is the shortest path subgraph of G_σ, we can restate Lemma 6.6 from [4] as follows.

▶ **Lemma 20.** *For any $i \in [k]$, $|C_{\sigma,i+1}| \leq 2 \cdot |C_{\sigma,i}|$.*

Reader may note that the proof of the above lemma in [4] crucially relies on Assumption 12.

▶ **Lemma 21.** $|\mathcal{E}(t)| \leq e(k-1)!2^k$.

Proof. In our algorithm for each $\sigma \in \{-1, 0, \cdots, k\}^k$ we compute the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k}$ and add $|C_{\sigma,k+1}|$ many edges in the set $\mathcal{E}(t)$ only if for all $i' < i$, $\sigma(i') < k - i + i' - 1$ where $i = \min\{j \mid \sigma(j) = -1\}$; otherwise we do not compute anything. So the total number of σ for which we add edges in $\mathcal{E}(t)$ is bounded by

$$1+(k-1)+(k-1)(k-2)+\cdots+(k-1)! = (k-1)![1/0!+1/1!+\cdots+1/(k-1)!] \leq e\cdot(k-1)!.$$

Now by applying Lemma 20, we get that for each such σ, $|C_{\sigma,k+1}| \leq 2^k$ (since $|C_{\sigma,1}| = 1$) and this proves the claimed bound. ◀

Complexity analysis. Here we just mention that since by Lemma 11 finding k-WTSS requires n rounds where in each round we find k-WTSS(v) for some $v \in V(G)$, computing k-WTSS takes total $O(k^k m^2 n)$ time.

6 Lower Bound Results for the Size of k-WTSS

Theorem 3 already provides a lower bound on the size of k-WTSS with the restriction same as that considered in our k-WTSS construction. In this section we show that the size of k-WTSS of a graph can be of size at least $\Omega(n^2)$ even for $k = 1$ if we allow either the weight function or the increment function to be rational valued. We formally state the lower bounds in the following two theorems, proofs of which can be found in the full version [13].

▶ **Theorem 22.** *If weight of an edge can be any rational value, then for every $n \in \mathbb{N}$, there exists a directed graph with n vertices whose 1-WTSS must contain $c \cdot n^2$ many edges for some constant $c > 0$.*

▶ **Theorem 23.** *If it is allowed to increase the weight of the edges by any rational value, then for every $n \in \mathbb{N}$, there exists a directed graph with n vertices whose 1-WTSS must contain $c \cdot n^2$ many edges for some constant $c > 0$.*

▶ Remark. We emphasize that all the lower bound results in the above section hold for undirected graphs also. Moreover, exactly the same graphs without any direction will serve the purpose.

References

1. Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 351–361, 2016.
2. Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 568–576, 2017.
3. Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single source fault tolerant shortest path. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1901–1915, 2018.
4. Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 509–518, 2016.

5 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (alpha, beta)-spanners and purely additive spanners. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 672–681, 2005.

6 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 101–110, 2009.

7 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, pages 167–178, 2015.

8 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Proceedings*, pages 137–148, 2014.

9 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 18:1–18:14, 2016.

10 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 73:1–73:14, 2017.

11 Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and (μ, α)-spanners. *Theor. Comput. Sci.*, 580:94–100, 2015.

12 Luciana S. Buriol, Mauricio G. C. Resende, and Mikkel Thorup. Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing*, 20(2):191–204, 2008.

13 Diptarka Chakraborty and Debarati Das. Near optimal sized weight tolerant subgraph for single source shortest path. *CoRR*, abs/1707.04867, 2017.

14 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 435–444, 2009.

15 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 130:1–130:13, 2016.

16 Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

17 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 260–267, 2001.

18 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

19 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011*, pages 169–178, 2011.

20 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 506–515, 2009.

21 D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 2010.

22 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 748–757, 2012.

23 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 127:1–127:15, 2017.

24 Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

25 Tamás Lukovszki. New results of fault tolerant geometric spanners. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Proceedings*, pages 193–204, 1999.

26 Merav Parter. Vertex fault tolerant additive spanners. In *Distributed Computing - 28th International Symposium, DISC 2014, Proceedings*, pages 167–181, 2014.

27 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 481–490, 2015.

28 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, pages 779–790, 2013.

29 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092, 2014.

30 G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest path problem. *Journal of Algorithms*, 21:267–305, 1996.

31 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

32 Hans Rohnert. A dynamization of the all pairs least cost path problem. In *STACS 85, 2nd Symposium of Theoretical Aspects of Computer Science, Proceedings*, pages 279–286, 1985.

33 Mohammadreza Saeedmanesh and Nikolas Geroliminis. Dynamic clustering and propagation of congestion in heterogeneously congested urban traffic networks. *Transportation Research Part B: Methodological*, 105(Supplement C):193–211, 2017.

34 Maxim Teslenko and Elena Dubrova. An efficient algorithm for finding double-vertex dominators in circuit graphs. In *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005)*, pages 406–411, 2005.

35 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013.

36 Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1337–1346, 2011.

37 David P. Woodruff. Additive spanners in nearly quadratic time. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010*, pages 463–474, 2010.

An Improved Algorithm for Incremental DFS Tree in Undirected Graphs

Lijie Chen
Massachusetts Institute of Technology
lijieche@mit.edu

Ran Duan[1]
Tsinghua University
duanran@mail.tsinghua.edu.cn

Ruosong Wang
Carnegie Mellon University
ruosongw@andrew.cmu.edu

Hanrui Zhang
Duke University
hrzhang@cs.duke.edu

Tianyi Zhang
Tsinghua University
tianyi-z16@mails.tsinghua.edu.cn

Abstract

Depth first search (DFS) tree is one of the most well-known data structures for designing efficient graph algorithms. Given an undirected graph $G = (V, E)$ with n vertices and m edges, the textbook algorithm takes $O(n + m)$ time to construct a DFS tree. In this paper, we study the problem of maintaining a DFS tree when the graph is undergoing incremental updates. Formally, we show:

> Given an arbitrary online sequence of edge or vertex insertions, there is an algorithm that reports a DFS tree in $O(n)$ worst case time per operation, and requires $O\left(\min\{m \log n, n^2\}\right)$ preprocessing time.

Our result improves the previous $O(n \log^3 n)$ worst case update time algorithm by Baswana et al. [1] and the $O(n \log n)$ time by Nakamura and Sadakane [15], and matches the trivial $\Omega(n)$ lower bound when it is required to explicitly output a DFS tree.

Our result builds on the framework introduced in the breakthrough work by Baswana et al. [1], together with a novel use of a tree-partition lemma by Duan and Zhang [9], and the celebrated fractional cascading technique by Chazelle and Guibas [6, 7].

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases DFS tree, fractional cascading, fully dynamic algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.16

Acknowledgements The authors would like to thank Shahbaz Khan, Kasper Green Larsen and Seth Pettie for many helpful discussions, and the anonymous reviewer for pointing out an issue in an earlier version of this paper.

[1] R. Duan is supported by a China Youth 1000-Talent grant.

1 Introduction

Depth First Search (DFS) is one of the most renowned graph traversal techniques. After Tarjan's seminal work [21], it demonstrates its power by leading to efficient algorithms to many fundamental graph problems, e.g., biconnected components, strongly connected components, topological sorting, bipartite matching, dominators in directed graph and planarity testing.

Real world applications often deal with graphs that keep changing with time. Therefore it is natural to study the dynamic version of graph problems, where there is an online sequence of updates on the graph, and the algorithm aims to maintain the solution of the studied graph problem efficiently after seeing each update. The last two decades have witnessed a surge of research in this area, like connectivity [10, 12, 13, 14], reachability [18, 20], shortest path [8, 19], bipartite matching [3, 16], and min-cut [22].

We consider the dynamic maintenance of DFS trees in undirected graphs. As observed by Baswana et al. [1] and Nakamura and Sadakane [15], the *incremental* setting, where edges/vertices are added but never deleted from the graph, is arguably easier than the *fully dynamic* setting where both kinds of updates can happen – in fact, they provide algorithms for incremental DFS with $\tilde{O}(n)$ worst case update time, which is close to the trivial $\Omega(n)$ lower bound when it is required to explicitly report a DFS tree after each update. ***So, is there an algorithm that requires nearly linear preprocessing time and space, and reports a DFS tree after each incremental update in*** $O(n)$ ***time?*** In this paper, we study the problem of maintaining a DFS tree in the incremental setting, and give an affirmative answer to this question.

1.1 Previous works on dynamic DFS

Despite the significant role of DFS tree in static algorithms, there is limited progress on maintaining a DFS tree in the *dynamic* setting.

Many previous works focus on the *total time* of the algorithm for any arbitrary updates. Franciosa et al. [11] designed an incremental algorithm for maintaining a DFS tree in a DAG from a given source, with $O(mn)$ total time for an arbitrary sequence of edge insertions; Baswana and Choudhary [2] designed a decremental algorithm for maintaining a DFS tree in a DAG with expected $O(mn \log n)$ total time. For undirected graphs, Baswana and Khan [4] designed an incremental algorithm for maintaining a DFS tree with $O(n^2)$ total time.

These algorithms used to be the only results known for the dynamic DFS tree problem. However, none of these existing algorithms, despite that they are designed for only a partially dynamic environment, achieves a worst case bound of $o(m)$ on the update time.

That barrier is overcome in the recent breakthrough work of Baswana et al. [1], they provide, for undirected graphs, a fully dynamic algorithm with worst case $O(\sqrt{mn} \log^{2.5} n)$ update time, and an incremental algorithm with worst case $O(n \log^3 n)$ update time. Due to the rich information in a DFS tree, their results directly imply faster worst case fully dynamic algorithms for subgraph connectivity, biconnectivity and 2-edge connectivity.

The results of Baswana et al. [1] suggest a promising way to further improve the worst case update time or space consumption for those fully dynamic algorithms by designing better dynamic algorithms for maintaining a DFS tree. In particular, based on the framework by Baswana et al. [1], Nakamura and Sadakane [15] propose an algorithm which takes $O(\sqrt{mn} \log^{1.75} n / \sqrt{\log \log n})$ time per update in the fully dynamic setting and $O(n \log n)$ time in the incremental setting, and $O(m \log n)$ bits of space.

1.2 Our results

In this paper, following the approach of [1], we improve the update time for the incremental setting, also studied in [1], by combining a better data structure, a novel tree-partition lemma by Duan and Zhang [9] and the fractional-cascading technique by Chazelle and Guibas [6, 7].

For any set U of incremental updates (insertion of a vertex/an edge), we let $G+U$ denote the graph obtained by applying the updates in U to the graph G. Our results build on the following main theorem.

▶ **Theorem 1.** *There is a data structure with $O(\min\{m \log n, n^2\})$ size, and can be built in $O(\min\{m \log n, n^2\})$ time, such that given a set U of k insertions, a DFS tree of $G+U$ can be reported in $O(n+k)$ time.*

By the above theorem combined with a de-amortization trick in [1], we establish the following corollary for maintaining a DFS tree in an undirected graph with incremental updates.

▶ **Corollary 2 (Incremental DFS tree).** *Given a sequence of online edge/vertex insertions, a DFS tree can be maintained in $O(n)$ worst case time per insertion.*

1.3 Organization of the Paper

In Section 2 we introduce frequently used notations and review two building blocks of our algorithm – the tree partition structure [9] and the fractional cascading technique [6, 7]. In Section 3 and Section 4, we study a batched version of the incremental setting, where all incremental updates are given at once, after which a single DFS tree is to be reported. After that, by a standard de-amortization technique, our algorithm for the batched setting directly implies the efficient algorithm for the incremental setting stated in Corollary 2.

2 Preliminaries

Let $G = (V, E)$ denote the original graph, T a corresponding DFS tree rooted at a special vertex $r \in V$, and U a set of inserted vertices and edges. We first introduce necessary notations.

- $T(x)$: The subtree of T rooted at x.
- $path(x, y)$: The path from x to y in T.
- $par(v)$: The parent of v in T.
- $N(x)$: The adjacency list of x in G.
- $L(x)$: The reduced adjacency list for vertex x, which is maintained during the algorithm.
- T^*: The newly generated DFS tree after the batch insertion U.
- $par^*(v)$: The parent of v in T^*.

Our algorithm uses a tree partition lemma in [9] and the famous fractional cascading structure in [6, 7], which are summarized as the following two lemmas.

▶ **Lemma 3** (Tree partition structure [9]). *Given a rooted tree T and any integer parameter k such that $2 \le k \le n = |V(T)|$, we can mark a subset of vertices of no more than $3n/k - 5$, such that after removing all marked vertices, the tree T is partitioned into sub-trees of size at most k. Also, the marked vertex subset can be computed in $O(n \log n)$ time.*

Algorithm 1: BatchInsert

Data: a DFS tree T of G, set of insertions U
Result: a DFS tree T^* of $G + U$
Add each inserted vertex v into T, set $par(v) = r$;
Initialize $L(v)$ to be $\emptyset$ for each v;
Add each inserted edge (u, v) to $L(u)$ and $L(v)$;
Call DFS(r);

Algorithm 2: DFS

Data: a DFS tree T of G, the entering vertex v
Result: a partial DFS tree
Let $u = v$;
while *$par(u)$ is not visited* **do**
    Let $u = par(u)$;
Mark $path(u, v)$ to be visited;
Let $(w_1, \ldots, w_t) = path(u, v)$;
for $i \in [t]$ **do**
    if $i \neq t$ **then**
        Let $par^*(w_i) = w_{i+1}$;
    for *child x of w_i in T except w_{i+1}* **do**
        Let $(y, z) = Q(T(x), u, v)$, where $y \in path(u, v)$;
        Add z into $L(y)$;
for $i \in [t]$ **do**
    for $x \in L(w_i)$ **do**
        if *x is not visited* **then**
            Let $par^*(x) = w_i$;
            Call DFS(x);

▶ **Lemma 4** (Fractional cascading [6, 7]). *Given k sorted arrays $\{A_i\}_{i\in[k]}$ of integers with total size $\sum_{i=1}^{k} |A_i| = m$. There exists a data structure which can be built in $O(m)$ time and using $O(m)$ space, such that for any integer x, the successors of x in all A_i's can be found in $O(k + \log m)$ time.*

3 Handling batch insertions

In this section, we study the dynamic DFS tree problem in the batch insertion setting. The goal of this section is to prove Theorem 1. Our algorithm basically follows the same framework for fully dynamic DFS proposed in [1]. Since we are only interested in the batch insertion setting, we can moderately simplify their algorithms by directly pruning those details unrelated to insertions, as described in pseudo-code BatchInsert (Algorithm 1) and DFS (Algorithm 2).

In Algorithm BatchInsert, we first attach each inserted vertex to the super root r, and pretend it has been there since the very beginning. Then only edge insertions are to be considered. All inserted edges are added into the reduced adjacency lists of corresponding

vertices. We then use DFS to traverse the graph starting from r based on T, L, and build the new DFS tree while traversing the entire graph and updating the reduced adjacency lists.

In Algorithm DFS, the new DFS tree is built in a recursive fashion. Every time we enter an untouched subtree, say $T(u)$, from vertex $v \in T(u)$, we change the root of $T(u)$ to v and go through $path(v, u)$; i.e., we wish to reverse the order of $path(u, v)$ in T^*. One crucial step behind this operation is that we need to find a new root for each subtree $T(w)$ originally hanging on $path(u, v)$. The following lemma tells us where the $T(w)$ should be rerooted on $path(u, v)$ in T^*.

▶ **Lemma 5** ([1]). *Let T^* be a partially constructed DFS tree, v the current vertex being visited, w an (not necessarily proper) ancestor of v in tree T^*, and C a connected component of the subgraph induced by unvisited vertices. If there are two edges e and e' from C incident on v and w, then it is sufficient to consider only e during the rest of the DFS traversal.*

Let $Q(T(w), u, v)$ be the edge between the highest vertex on $path(u, v)$ incident to a vertex in subtree $T(w)$, and the corresponding vertex in $T(w)$. $Q(T(w), u, v)$ is defined to be Null if such an edge does not exist. By Lemma 5, it suffices to ignore all other edges but just keep the edge returned by $Q(T(w), u, v)$; this is because we have reversed the order of $path(u, v)$ in T^* and thus $Q(T(w), u, v)$ connects to the lowest possible position in T^*. Hence $T(w)$ should be rerooted at $Q(T(w), u, v)$.

Denote (x, y) to be the edge returned by $Q(T(w), u, v)$ where $x \in path(u, v)$, and then we add y into $L(x)$. After finding an appropriate entering edge for each hanging subtree, we process each vertex $v \in path(u, v)$ in ascending order of depth (with respect to tree T). For every unvisited $w \in L(v)$, we set $par^*(w) = v$, and recursively call DFS(w).

▶ **Theorem 6.** *BatchInsert correctly reports a feasible DFS tree T^* of graph $G + U$.*

Proof. We argue that in a single call DFS(v), where u is the highest unvisited ancestor of v, every unvisited (at the moment of being enumerated) subtree $T(w)$ hanging from $path(u, v)$, as well as every vertex on $path(u, v)$ except v, will be assigned an appropriate parent such that these parent-child relationships constitute a DFS tree of G at the termination of BatchInsert. When the traversal reaches v, the entire $T(u)$ is untouched, or else u would have been marked by a previous visit to some vertex in $T(u)$. We could therefore choose to go through $path(v, u)$ to reach u first. By Lemma 5, if a subtree $T(w)$ is reached from some vertex on $path(u, v)$, it suffices to consider only the edge $Q(T(w), u, v)$. After adding the query results of all hanging subtrees into the adjacency lists of vertices on $path(u, v)$, every hanging subtree visited from some vertex x on $path(u, v)$ should be visited in a correct way through edges in $L(x)$ solely. Since every vertex will eventually be assigned a parent, BatchInsert does report a feasible DFS tree of graph $G + U$. ◀

For now we have not discussed how to implement $Q(T(w), u, v)$ and the above algorithm only assumes blackbox queries to $Q(T(\cdot), \cdot, \cdot)$. The remaining problem is to devise a data structure $\mathcal{D}$ to answer all the queries demanded by Algorithm DFS in $O(n)$ total time. We will show in the next section that there exists a data structure $\mathcal{D}$ with the desired performance, which is stated as the following lemma.

▶ **Lemma 7.** *There exists a data structure $\mathcal{D}$ with preprocessing time $O\left(\min\{m \log n, n^2\}\right)$ time and space complexity $O\left(\min\{m \log n, n^2\}\right)$ that can answer all queries $Q(T(w), x, y)$ in a single run of BatchInsert in $O(n)$ time.*

Proof of Theorem 1. By Lemma 7, the total time required to answer queries is $O(n)$. The total size of reduced adjacency lists is bounded by $O(n + |U|)$, composed by $O(|U|)$ edges

added in BatchInsert and $O(n)$ added during DFS. Thus, the total time complexity of BatchInsert is $O(n + |U|)$.

During preprocessing, we use depth first search on G to get the initial DFS tree T, and build $\mathcal{D}$ in time $O\left(\min\{m \log n, n^2\}\right)$. The total time for preprocessing is $O\left(\min\{m \log n, n^2\}\right)$. ◀

4 Dealing with queries in BatchInsert

In this section we prove Lemma 7. Once this goal is achieved, the overall time complexity of batch insertion taken by Algorithm BatchInsert would be $O(n + |U|)$.

In the following part of this section, we will first devise a data structure in Section 4.1, that answers any single query $Q(T(w), u, v)$ in $O(\log n)$ time, which would be useful in other parts of the algorithm. We will then present another simple data structure in Section 4.2, which requires $O(n^2)$ preprocessing time and $O(n^2)$ space and answers each query in $O(1)$ time. Finally, we propose a more sophisticated data structure in Section 4.3, which requires $O(m \log n)$ preprocessing time and $O(m \log n)$ space and answers all queries $Q(T(w), x, y)$ in a single run of BatchInsert in $O(n)$ time. Hence, we can always have an algorithm that handles a batch insertion U in $O(n + |U|)$ time using $O(\min\{m \log n, n^2\})$ preprocessing time and $O(\min\{m \log n, n^2\})$ space, thus proving Theorem 1. We can then prove Corollary 2 using the following standard de-amortization argument.

▶ **Lemma 8.** *(Lemma 6.1 in [1]) Let $\mathcal{D}$ be a data structure that can be used to report the solution of a graph problem after a set of U updates on an input graph G. If $\mathcal{D}$ can be initialized in $O(f)$ time and the solution for graph $G+U$ can be reported in $O(h+|U| \times g)$ time, then $\mathcal{D}$ can be modified to report the solution after every update in worst-case $O\left(\sqrt{fg} + h\right)$ update time after spending $O(f)$ time in initialization, given that $\sqrt{f/g} \leq n$.*

Proof of Corollary 2. Taking $f = \min\{m \log n, n^2\}$, $g = 1$, $h = n$ and directly applying the above lemma will yield the desired result. ◀

4.1 Answering a single query in $O(\log n)$ time

We show in this subsection that the query $Q(T(\cdot), \cdot, \cdot)$ can be reduced efficiently to the range successor query (see, e.g., [17], for the definition of range successor query), and show how to answer the range successor query, and thus any individual query $Q(T(\cdot), \cdot, \cdot)$, in $O(\log n)$ time.

To deal with a query $Q(T(w), x, y)$, first note that since T is a DFS tree, all edges not in T but in the original graph G must be ancestor-descendant edges. Querying edges between $T(w)$ and $path(x, y)$ where x is an ancestor of y and $T(w)$ is hanging from $path(x, y)$ is therefore equivalent to querying edges between $T(w)$ and $path(x, par(w))$, i.e., $Q(T(w), x, y) = Q(T(w), x, par(w))$. From now on, we will consider queries of the latter form only.

Consider the DFS sequence of T, where the i-th element is the i-th vertex reached during the DFS on T. Note that every subtree $T(w)$ corresponds to an interval in the DFS sequence. Denote the index of vertex v in the DFS sequence by $first(v)$, and the index of the last vertex in $T(v)$ by $last(v)$. During the preprocessing, we build a 2D point set S. For each edge $(u, v) \in E$, we add a point $p = (first(u), first(v))$ into S. Notice that for each point $p \in S$, there exists exactly one edge (u, v) associated with p. Finally we build a 2D range tree [6, 7] on point set S with $O(m \log n)$ space and $O(m \log n)$ preprocessing time.

To answer an arbitrary query $Q(T(w), x, par(w))$, we query the point with minimum x-coordinate lying in the rectangle $\Omega = [first(x), first(w) - 1] \times [first(w), last(w)]$. If no

such point exists, we return Null for $Q(T(w), x, par(w))$. Otherwise we return the edge corresponding to the point with minimum x-coordinate.

Now we prove the correctness of our approach.

- If our method returns Null, $Q(T(w), x, par(w))$ must equal Null. Otherwise, suppose $Q(T(w), x, par(w)) = (u, v)$. Noticing that $(first(u), first(v))$ is in Ω, it means our method will not return Null in that case.
- If our method does not return Null, denote (u', v') to be the edge returned by our method. We can deduce from the query rectangle that $u' \in T(x) \backslash T(w)$ and $v' \in T(w)$. Thus, $Q(T(w), x, par(w)) \neq$ Null. Suppose $Q(T(w), x, par(w)) = (u, v)$. Notice that $(first(u), first(v))$ is in Ω, which means $first(u') \leq first(u)$. If $u' = u$, then our method returns a feasible solution. Otherwise, from the fact that $first(u') < first(u)$, we know that u' is an ancestor of u, which contradicts the definition of $Q(T(w), x, par(w))$.

4.2 An $O(n^2)$-space data structure

In this subsection we propose a data structure with quadratic preprocessing time and space complexity that answers any $Q(T(\cdot), \cdot, \cdot)$ in constant time.

Since we allow quadratic space, it suffices to precompute and store answers to all possible queries $Q(T(w), u, par(w))$. For preprocessing, we enumerate each subtree $T(w)$, and fix the lower end of the path to be $v = par(w)$ while we let the upper end u go upward from v by one vertex at a time to calculate $Q(T(w), u, par(w))$ incrementally, in order to answer all queries of the form $Q(T(w), \cdot, par(w))$ in $O(n)$ total time.

As u goes upward on tree T, we need to find an edge, if existent, from u to $T(w)$ in $O(1)$ time. To to this, we pre-compute, using dynamic programming, the answers for all possible w's as an independent task in $O(n)$ total time, for each fixed $u \in V$. Prepare an array $A_u[\cdot]$ indexed by all descendants w of u which is initialised with all Null's. Start listing all descendants w incident to u. For each such w, we enumerate its ancestors v below u in descending order in terms of depth and reset $A_u[v] = (u, w)$, and this enumeration gets halted when we meet for the first time an ancestor v with $A_u[v]$ already $\neq$ Null. It is clear that u is connected to $T(w)$ iff $A_u[w] \neq$ Null, since for each descendant w where u is incident to some vertices in $T(w)$, $A_u[w]$ is reset when the algorithm lists the first neighbour of u in $T(w)$. The total time of this procedure is clearly $O(n)$ since every $A_u[v]$ is manipulated for at most once.

▶ **Lemma 9.** *The preprocessing time and query time of the above data structure are $O(n^2)$ and $O(1)$ respectively.*

Proof. The array A_u can be built for each vertex u in total time $O(n^2)$. For each subtree $T(w)$, we go up the path from w to the root r, and spend $O(1)$ time for each vertex u on $path(r, w)$ to get the answer for $Q(T(w), u, par(w))$. There are at most n vertices on $path(r, w)$, so the time needed for a single subtree is $O(n)$, and that needed for all subtrees is $n \cdot O(n) = O(n^2)$ in total. On the other hand, for each query, we simply look it up and answer in $O(1)$ time. Hence we conclude that the preprocessing time and query time are $O(n^2)$ and $O(1)$ respectively. ◀

4.3 An $O(m \log n)$-space data structure

Observe that in BatchInsert (and DFS), a bunch of queries $\{Q(T(w_i), x, y)\}$ are always made simultaneously, where $\{T(w_i)\}$ is the set of subtrees hanging from $path(x, y)$. We may

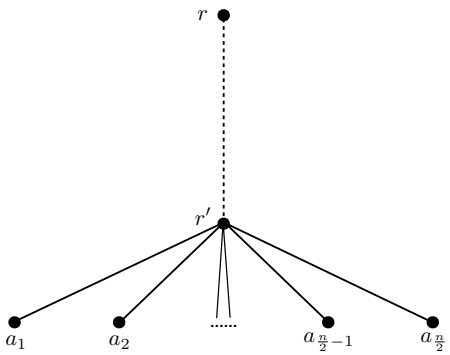


Figure 1 In this example, if we stick to the 2D-range-based data structure introduced before, then computing all $Q(T(a_i), r, r')$ would take as much as $O(n \log n)$ time.

therefore answer all queries for a path in one pass, instead of answering them one by one. By doing so we confront two types of hard queries.

First consider an example where the original DFS tree T is a chain L where a_1 is the root of L and for $1 \leq i \leq n-1$, a_{i+1} is the unique child of a_i. When we invoke $\mathsf{DFS}(a_1)$ on L, $path(u, v)$ is the single node a_1. Thus, we will call $Q(T(a_2), a_1, a_1)$ and add the returned edge into $L(a_1)$. Supposing there are no back-edges in this graph, the answer of $Q(T(a_2), a_1, a_1)$ will be the edge (a_1, a_2). Therefore, we will recursively call the $\mathsf{DFS}(a_2)$ on the chain (a_2, a_n). Following further steps of DFS, we can see that we will call the query $Q(T(w), x, y)$ for $\Omega(n)$ times. For the rest of this subsection, we will show that we can deal with this example in linear time. The idea is to answer queries involving short paths in constant time. For instance, in the example shown above, $path(u, v)$ always has constant length. We show that when the length of $path(u, v)$ is smaller than $2 \log n$, it is affordable to preprocess all the answers to queries of this kind in $O(m \log n)$ time and $O(n \log n)$ space.

The second example we considered is given as Figure 1. In this tree, the original root is r. Suppose the distance between r and r' is $n/2$. When we invoke $\mathsf{DFS}(r')$, $path(u, v)$ the path from r to r'. Thus, we will call $T(a_1, r, r')$, $T(a_2, r, r')$, ..., $T(a_{n-2}, r, r')$, which means we make $\Omega(n)$ queries. In order to deal with this example in linear time, the main idea is using fractional cascading to answer all queries $Q(T(w), x, y)$ with a fixed $path(u, v)$, for all subtrees $T(w)$ with small size.

In the examples shown above, all subtrees cut off $path(u, v)$ have constant size and thus the total time complexity for this example is $O(n)$. We will finally show that, by combining the two techniques mentioned above, it is enough to answer all queries $Q(T(w), x, y)$ in linear time, thus proving Lemma 7.

Data structure

The data structure consists of the following parts.

- **(i)** Build the 2D-range successor data structure that answers any $Q(T(\cdot), \cdot, \cdot)$ in $O(\log n)$ time.
- **(ii)** For each ancestor-descendent pair (u, v) such that u is at most $2 \log n$ hops above v, precompute and store the value of $Q(T(v), u, par(v))$.

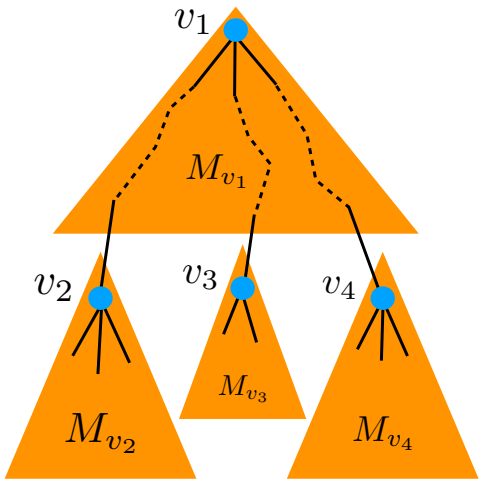


Figure 2 In this example, each blue node represents a vertex $v_i (1 \le i \le 4)$ from set M, and M_{v_i}'s are drawn as yellow triangles. For each triangle, a fractional cascading data structure is built on adjacency lists of all vertices inside.

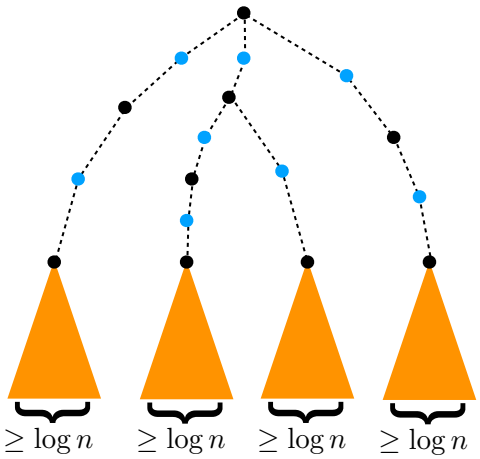


Figure 3 In this picture, sets M and $X \cup \{r\}$ are drawn as blue nodes and black nodes respectively, and each yellow triangle is a subtree rooted at a leaf of $T[X]$, which has size $\ge \log n$. Note that every ancestor-descendent tree path between two black nodes contains a blue node.

(iii) Apply Lemma 3 with parameter $k = \log n$ and obtain a marked set of size $O(n/\log n)$. Let M be the set of all marked vertices x such that $|T(x)| \ge \log n$. For every $v \notin M$, let $anc_v \in M$ be the nearest ancestor of v in set M.
Next we build a fractional cascading data structure for each $u \in M$ in the following way. Let M_u be the set of all vertices in $T(u)$ whose tree paths to u do not intersect any other vertices $u' \ne u$ from M, namely $M_u = \{v \mid anc_v = u\}$; see Figure 2 for an example. Then, apply Lemma 4 on all $N(v), v \in M_u$ where $N(v)$ is treated as sorted array in an ascending order with respect to depth of the edge endpoint opposite to v; this would build a fractional cascading data structure that, for any query encoded as a $w \in V$, answers for every $v \in M_u$ its highest neighbour below vertex w in total time $O(|M_u| + \log n)$.

Here is a structural property of M that will be used when answering queries.

▶ **Lemma 10.** *For any ancestor-descendent pair (u, v), if $path(u, v) \cap M = \emptyset$, then $path(u, v)$ has $\le 2 \log n$ hops.*

Proof. Suppose otherwise. By definition of marked vertices there exists a marked vertex $w \in path(u, v)$ that is $\le \log n$ hops below u. Then since $path(u, v)$ has $> 2 \log n$ many hops, it must be $T(w) \ge \log n$ which leads to $w \in M$, contradicting $path(u, v) \cap M = \emptyset$. ◀

Preprocessing time

First of all, for part (i), as discussed in a previous subsection, 2D-range successor data structure takes time $O(m \log n)$ to initialize. Secondly, for part (iii), on the one hand by Lemma 3 computing a tree partition takes time $O(n \log n)$; on the other hand, by Lemma 4, initializing the fractional cascading with respect to $u \in M$ costs $O(\sum_{v \in M_u} |N(v)|)$ time.

Since, by definition of M_u, each $v \in V$ is contained in at most one $M_u, u \in M$, the overall time induced by this part would be $O(\sum_{u \in M} \sum_{v \in M_u} |N(v)|) = O(m)$.

Preprocessing part (ii) requires a bit of cautions. The procedure consists of two steps.

(1) For every ancestor-descendent pair (u, v) such that u is at most $2 \log n$ hops above v, we mark (u, v) if u is incident to $T(v)$.
Here goes the algorithm: for every edge $(u, w) \in E$ (u being the ancestor), let $z \in path(u, w)$ be the vertex which is $2 \log n$ hops below u (if $path(u, w)$ has less than $2 \log n$ hops, then simply let $z = w$); note that this z can be found in constant time using the level-ancestor data structure [5] which can be initialized in $O(n)$ time. Then, for every vertex $v \in path(u, z)$, we associate the pair (u, v) with edge (u, w); if a vertex pair is associated with more than one edge, we only keep an arbitrary one. The total running time of this procedure is $O(m \log n)$ since each edge (u, w) takes up $O(\log n)$ time.

(2) Next, for each $v \in V$, we compute all entries $Q(T(v), u, par(v))$ required by (ii) in an incremental manner. Let $u_1, u_2, \cdots, u_{2 \log n}$ be the nearest $2 \log n$ ancestors of v sorted in descending order with respect to depth, and then we directly solve the recursion

$$Q(T(v), u_{i+1}, par(v)) = \begin{cases} Q(T(v), u_i, par(v)) & (u_{i+1}, v) \text{ is not associated with any edge} \\ (u_{i+1}, w) & (u_{i+1}, v) \text{ is associated with edge } (u_{i+1}, w) \end{cases}$$

for all $0 \leq i < 2 \log n$ in $O(\log n)$ time. Note that no $Q(T(v), u_{i+1}, par(v))$ is an undefined value since (u_1, v) is always associated with an edge, say (u_1, v) itself. The total running time would thus be $O(n \log n)$.

Summing up (i)(ii)(iii), the preprocessing time is bounded by $O(m \log n)$.

Query algorithm and total running time

We show how to utilize the above data structures (i)(ii)(iii) to implement $Q(T(\cdot), \cdot, \cdot)$ on line 9-11 in Algorithm DFS such that the overall time complexity induced by this part throughout a single execution of Algorithm BatchInsert is bounded by $O(n)$.

Let us say we are given $(w_1, w_2, \cdots, w_t) = path(u, v)$ and we need to compute $Q(T(x), u, v)$ for every subtree $T(x)$ that is hanging on $path(u, v)$. There are three cases to discuss.

(1) If $path(u, v) \cap M = \emptyset$, by Lemma 10 we claim $path(u, v)$ has at most $2 \log n$ hops, and then we can directly retrieve the answer of $Q(T(x), u, v)$ from precomputed entries of (ii), each taking constant query time.

(2) Second, consider the case where $path(u, v) \cap M \neq \emptyset$. Let $s_1, s_2, \cdots, s_l, l \geq 1$ be the consecutive sequence (in ascending order with respect to depth in tree T) of all vertices from M that are on $path(u, v)$. For those subtrees $T(x)$ that are hanging on $path(u, par(s_1))$, we can directly retrieve the value of $Q(T(x), u, par(x))$ from (ii) in constant time, as by Lemma 10 $path(u, par(s_1))$ has at most $2 \log n$ hops.

(3) Third, we turn to study the value of $Q(T(x), u, par(x))$ when $par(x)$ belongs to a $path(s_i, par(s_{i+1})), i < l$ or $path(s_l, v)$. The algorithm is two-fold.

(a) First, we make a query of u to the fractional cascading data structure built at vertex s_i $(1 \leq i \leq l)$, namely part (iii), which would give us, for every descendent $y \in M_{s_i}$, the highest neighbour of y below u. Using this information we are able to derive the result of $Q(T(x), u, v)$ if $|T(x)| < \log n$, since in this case $T(x) \cap M = \emptyset$ and thus $T(x) \subseteq M_{s_i}$.
By Lemma 4 the total time of this procedure is $O(|M_{s_i}| + \log n)$.

(b) We are left to deal with cases where $|T(x)| \geq \log n$. In this case, we directly compute $Q(T(x), u, v)$ using the 2D-range successor built in (i) which takes $O(\log n)$ time.

Correctness of the query algorithm is self-evident. The total query time is analysed as follows. Throughout an execution of Algorithm BatchInsert, (1) and (2) contribute at most $O(n)$ time since each $T(x)$ is involved in at most one such query $Q(T(x), u, v)$ which takes constant time. As for (3)(a), since each marked vertex $s \in M$ lies in at most one such path $(w_1, w_2, \cdots, w_t) = path(u, v)$, the fractional cascading data structure associated with M_s is queried for at most once. Hence the total time of (3)(a) is $O(\sum_{s \in M}(|M_s| + \log n)) = O(n + |M| \log n) = O(n)$; the last equality holds by $|M| \leq O(n/\log n)$ due to Lemma 3.

Finally we analyse the total time taken by (3)(b). It suffices to upper-bound by $O(n/\log n)$ the total number of such x with the property that $|T(x)| \geq \log n$ and $path(u, par(x)) \cap M \neq \emptyset$. Let X be the set of all such x's.

▶ **Lemma 11.** *Suppose $x_1, x_2 \in X$ and x_1 is an ancestor of x_2 in tree T. Then $path(x_1, x_2) \cap M \neq \emptyset$.*

Proof. Suppose otherwise $path(x_1, x_2) \cap M = \emptyset$. Consider the time when query $Q(T(x_2), u, v)$ is made and let $path(u, v)$ be the path being visited by then. As $x_2 \in X$, by definition it must be $path(u, par(x_2)) \cap M \neq \emptyset$. Therefore, $path(u, x_2)$ is a strict extension of $path(x_1, x_2)$, and thus $x_1, par(x_1) \in path(u, x_2)$, which means x_1 and $par(x_1)$ become visited in the same invocation of Algorithm DFS. This is a contradiction since for any query of form $Q(T(x_1), \cdot, \cdot)$ to be made, by then $par(x_1)$ should be tagged "visited" while x_1 is not. ◀

Now we prove $|X| = O(n/\log n)$. Build a tree $T[X]$ on vertices $X \cup \{r\}$ in the natural way: for each $x \in X$, let its parent in $T[X]$ be x's nearest ancestor in $X \cup \{r\}$. Because of

$$|X| < 2\#\text{leaves of } T[X] + \#\text{vertices with a unique child in } T[X]$$

it suffices to bound the two terms on the right-hand side: on the one hand, the number of leaves of $T[X]$ is at most $n/\log n$ since for each leave x it has $|T(x)| \geq \log n$; on the other hand, for each $x \in T[X]$ with a unique child $y \in T[X]$, by Lemma 11 $path(x, y) \cap M \neq \emptyset$, and so we can charge this x to an arbitrary vertex in $path(x, y) \cap M$, which immediately bounds the total number of such x's by $|M| = O(n/\log n)$; see Figure 3 for an illustration. Overall, $|X| \leq O(n/\log n)$.

References

1. Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic dfs in undirected graphs: breaking the $O(m)$ barrier. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 730–739. SIAM, 2016.
2. Surender Baswana and Keerti Choudhary. On dynamic DFS tree in directed graphs. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 102–114. Springer, 2015.
3. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 383–392. IEEE, 2011.
4. Surender Baswana and Shahbaz Khan. Incremental algorithm for maintaining DFS tree for undirected graphs. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 138–149. Springer, 2014.

5 Michael A Bender and Martin Farach-Colton. The lca problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer, 2000.

6 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

7 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(1-4):163–191, 1986.

8 Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

9 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. *arXiv preprint arXiv:1605.04491*, 2016.

10 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)*, 44(5):669–696, 1997.

11 Paolo G Franciosa, Giorgio Gambosi, and Umberto Nanni. The incremental maintenance of a depth-first-search tree in directed acyclic graphs. *Information processing letters*, 61(2):113–120, 1997.

12 Monika R Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)*, 46(4):502–516, 1999.

13 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

14 Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142. Society for Industrial and Applied Mathematics, 2013.

15 Kengo Nakamura and Kunihiko Sadakane. A space-efficient algorithm for the dynamic dfs problem in undirected graphs. In *International Workshop on Algorithms and Computation*, pages 295–307. Springer, 2017.

16 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

17 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 271–282. Springer, 2012.

18 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008.

19 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM Journal on Computing*, 41(3):670–683, 2012.

20 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Foundations of Computer Science (FOCS), 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 509–517. IEEE, 2004.

21 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

22 Mikkel Thorup. Fully-dynamic min-cut. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing (STOC)*, pages 224–230. ACM, 2001.

Succinct Dynamic One-Dimensional Point Reporting

Hicham El-Zein
Cheriton School of Computer Science, University of Waterloo, Ontario, Canada N2L 3G1
helzein@uwaterloo.ca

J. Ian Munro[1]
Cheriton School of Computer Science, University of Waterloo, Ontario, Canada N2L 3G1
imunro@uwaterloo.ca

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Ontario, Canada N2L 3G1
ynekrich@uwaterloo.ca

Abstract

In this paper we present a succinct data structure for the dynamic one-dimensional range reporting problem. Given an interval $[a, b]$ for some $a, b \in [m]$, the range reporting query on an integer set $S \subseteq [m]$ asks for all points in $S \cap [a, b]$. We describe a data structure that answers reporting queries in optimal $O(k + 1)$ time, where k is the number of points in the answer, and supports updates in $O(\lg^{\varepsilon} m)$ expected time. Our data structure uses $\mathcal{B}(n, m) + o(\mathcal{B}(n, m))$ bits where $\mathcal{B}(n, m)$ is the minimum number of bits required to represent a set of size n from a universe of m elements. This is the first dynamic data structure for this problem that uses succinct space and achieves optimal query time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Succinct Data Structures, Range Searching, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.17

1 Introduction and Motivation

This paper studies the dynamic one-dimensional range reporting problem where the goal is to maintain (under insertion and deletion) a set of integers S from a universe of size m to answer range reporting queries efficiently: Given an interval $[a, b]$ for some $a, b \in [m]$, report all points in $S \cap [a, b]$. We note that the reporting query is equivalent to the query FindAny(a, b) which asks for an arbitrary point c in $S \cap [a, b]$: if the interval $[a, b]$ is not empty, we can recurse on $[a, c - 1]$ and $[c + 1, b]$ after obtaining any $c \in S \cap [a, b]$.

We study this problem in the succinct scenario. In the succinct setting the emphasis is on the space efficiency of the data structure. The goal is to design data structures that occupy optimal or almost-optimal space and at the same time achieve an efficient query cost. This area of research is of interest in theory and practice and is motivated by the need to store a large amount of data using the smallest space possible. In recent years there has been a surge of interest in succinct data structures for computational geometry [4, 2, 5, 10]. We refer the reader to the survey by Munro and Rao [11] and the book of Navarro [17] for a more in-depth coverage of succinct data structures.

[1] This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

Related Work. One-dimensional range reporting is a well studied problem. Miltersen et al. [13] presented a data structure for the static version of this problem that uses $O(n \lg m)$ words and answers queries in constant time per reported element. Alstrup et al. [1] later presented an improved data structure with the same query time that uses $O(n)$ words, i.e., $O(n \lg m)$ bits. Goswami et al. [7] presented a succinct data structure that further improved the space usage to $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits while preserving the query time where $\mathcal{B}(n,m) \approx n \lg (m/n)$ is the minimum number of bits required to represent a set of size n from a universe of m elements.

For the dynamic version of this problem Mortensen et al. [14] presented a data structure that uses a linear number of words and answers queries in $O(t_q)$ time and updates in expected $O(t_u)$ time where:

$t_q \geq \lg\lg\lg m$, $\lg\lg m / \lg\lg\lg m \leq t_u \leq \lg\lg m : \ t_u = O(\lg_{t_q} \lg m) + t_{pred}$,
or $t_q \leq \lg\lg\lg m$, $t_u \geq \lg\lg m : \ 2^{t_q} = O(\lg_{t_u} \lg m)$.

The most appealing point of this trade-off in the context of succinct data structures is when the query time is constant and the update time is $O(\lg^{\varepsilon} m)$ time for a fixed $\varepsilon > 0$.

Our Results. We start with some preliminaries in Section 2. In Section 3 we present a semi-dynamic succinct range reporting data structure that supports deletions in expected $O(\lg^{\varepsilon} m)$ time and queries in constant time. In Section 4 we present a fully-dynamic succinct range reporting data structure that supports updates in expected $O(\lg^{\varepsilon} m)$ time and queries in constant time. Our results depend on the ability to construct a static succinct one dimensional point reporting structure in $O(n \lg^{\varepsilon} m)$ time using $o(n)$ workspace. We defer the details of this construction to the end in Section 5 due to its technical nature.

2 Preliminaries

In this section we review some previous results that will be used in the rest of this paper.

2.1 One-Dimensional Point Reporting

First we review the data structure of Alstrup et al. [1] for static one-dimensional range reporting. We start by defining some notations. Let $x \oplus y$ denote the binary exclusive-or of x and y. Given a w-bit integer x let $x \downarrow i = x/2^i$ denote the rightmost w bits of the result of shifting x i bits to the right. Similarly let $x \uparrow i = x \cdot 2^i \mod 2^w$ denote the rightmost w bits of the result of shifting x i bits to the left. Finally, denote by $\operatorname{msb}(x)$ the position of the most significant bit (or leftmost one bit) of x.

Given a set of integers S the goal is to store S while supporting the query $\operatorname{FindAny}(a,b)$ which returns an element in $S \cap [a,b]$. Denote by T the classic binary tree with 2^w leaves where all leaves have depth w. The leaves are numbered $0, \ldots, 2^w - 1$ from left to right while the internal nodes are labeled in a manner similar to an implicit binary heap. The root is the first node, and the children of a node v are $2v$ and $2v+1$. As noted in [1] the d^{th} ancestor of v is $v \downarrow d$ and the lowest common ancestor of two leaves a and b is the $(1 + \operatorname{msb}(a \oplus b))^{\text{th}}$ ancestor of a or b. Thus the lowest common ancestor of two leaves can be computed in constant time.

Given a node $v \in T$ let $\operatorname{left}(v)$ and $\operatorname{right}(v)$ denote the left and right children of v, and let S_v denote the subset of S that is in the subtree rooted at v. A node v is branching if both $S_{\operatorname{left}(v)}$ and $S_{\operatorname{right}(v)}$ are not empty. To answer a query $\operatorname{FindAny}(a,b)$ it is sufficient to compute the lowest common ancestor v of a and b; when v is computed, either $\max S_{\operatorname{left}(v)}$ or $\min S_{\operatorname{right}(v)}$ is in $[a,b]$, or $[a,b]$ is empty. Thus by storing the values $\max S_{\operatorname{left}(v)}$ and

$\min S_{\text{right}(v)}$ for all nodes v with non-empty S_v in $O(nw)$ words, range reporting queries can be answered in constant time.

To improve the space Alstrup et al. [1] observe the following. Let v be the nearest branching ancestor of the lowest common ancestor of a and b, and let $v_l(v_r)$ be the nearest branching node in v's left(right) subtree if one exists, otherwise $v_l = v(v_r = v)$ if there is no branching node in v's left(right) subtree. Then either $\max S_{\text{left}(v_l)}$, $\min S_{\text{right}(v_l)}$, $\max S_{\text{left}(v_r)}$, or $\min S_{\text{right}(v_r)}$ is in $[a, b]$, or $[a, b]$ is empty. Thus they store a $O(n)$ word data structure that consists of:

$\boldsymbol{B}, \boldsymbol{D}$: vectors of size $O(n\sqrt{w}\lg w)$ bits that return the nearest branching ancestor of the nodes in T with non empty-subtrees.

$\boldsymbol{V}$: a vector storing for each branching node v the values $\max S_v$ and $\min S_v$, in addition to two pointers to the nearest branching nodes in the left and right subtrees of v.

For the full details we refer the reader to [1].

2.2 Tree Representation

In their paper Geary and Raman [6] present a succinct ordinal tree representation that answers level ancestor queries. In their tree representation the tree is partitioned into mini-trees of size $O(\lg^4 n)$, and then the mini-trees are partitioned into micro-trees of size $O(\lg n)$. Internally a node x is referred to by $\tau(x) = (\tau_1(x), \tau_2(x), \tau_3(x))$ where $\tau_1(x)$ is the id of x's mini-tree, $\tau_2(x)$ is the id of x's micro tree, and $\tau_3(x)$ is the id of x in its micro tree. If two nodes x and y are in the same micro tree μ then $\tau_1(x) = \tau_1(y) = p(\mu)$ where $p(\mu)$ is the id of the micro tree μ. Note that micro trees can intersect only at their roots, and if a node is in different micro trees (i.e. it is the root of several micro trees) it can have different τ names. That is, if a node x is a root of two different micro-trees μ_1 and μ_2, it will have two different τ names where in the first one $\tau_2(x) = p(\mu_1)$ and in the second $\tau_2(x) = p(\mu_2)$. Both names are valid and we can select any one of them.

Geary and Raman show how to compute the preorder number of x given $\tau(x)$ in constant time using an index of size $o(n)$ bits. This index can be constructed in $O(n)$ time using a workspace of $O(n)$ words. Given a tree T partitioned using the above scheme and a node $x \in T$ we denote by $\text{root}(x)$ the root of the mini-tree that x belongs to.

2.3 Sparse Arrays

We will use the following Theorem from [12]:

▶ **Theorem 1** ([12]). *There is an $(m, n, O(n))$-family of perfect hash functions $\mathcal{H}$ such that any hash function $h \in \mathcal{H}$ can be represented in $\Theta(n \lg\lg n)$ bits and evaluated in constant time for $m \leq 2^w$. The perfect hash function can be constructed in expected $O(n)$ time.*

As noted in [1] a corollary of the previous theorem is the following.

▶ **Corollary 2.** *A sparse array of size $m \geq n$ with n initialized entries that contain $b = \Omega(\lg\lg n)$ bits each can be stored using $O(nb)$ bits, so that any initialized entry can be accessed in $O(1)$ time. The expected preprocessing time of this data structure is $O(n)$.*

3 Semi-Dynamic Succinct One-Dimensional Point Reporting

Although Goswami et al. [7] presented a succinct data structure for one-dimensional range reporting, it is not clear what is the construction time of their data structure. In Section 5 we utilize succinct data structure techniques to improve the data structure in [1] so that it

uses $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits and can be constructed in $O(n \lg^{\varepsilon} m)$ time using $o(n)$ extra bits of space. The details are deferred to Section 5 due to their technical nature.

▶ **Theorem 3.** *There exists a succinct $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$-bit data structure that supports one-dimensional range reporting queries in $O(k+1)$ time where k is the number of points within the query. Additionally given the point set in sorted order, this data structure can be constructed in expected $O(n \lg^{\varepsilon} m)$ time using $o(n)$-bits workspace.*

The data structure for one-dimensional range reporting can be dynamized so that queries are supported in deterministic $O(k)$ time and updates in expected $O(\lg^{\varepsilon} m)$ time while the space usage is $O(n)$ words [14]. Our aim is to reduce the space to the information theoretic lower bound plus a lower order term. In this section we present a semi-dynamic succinct one-dimensional range reporting data structure that supports queries and deletions but does not support insertions.

Data Structure. We store the data structure from Theorem 3 and call it P. We divide the points into blocks of size $\lg^2 m$ and we store predecessor and successor data structures that can answer queries in each block independently using $o(\mathcal{B}(n,m))$ bits as described in [4]. We also store a dynamic data structure [14] D on the endpoints of each block. Furthermore, each block is divided into subblocks of size $\lg n/2$ and stores a dynamic data structure [14] D_i $(1 \leq i \leq n/\lg^2 m)$ on the ranks (within the block) of the endpoints of each subblock. We also store a compressed bit vector([8], Theorem 2) B of size n that indicates which points were deleted. Finally, we store a lookup table T that can report for any range the 0 bits in a bit vector of size $\lg n/2$.

Query. To report the points within an interval $[a,b]$ we query D on the interval. Then for each point reported with rank k we query the $(\lfloor k/2 \rfloor)^{\text{th}}$ and $(\lfloor k/2 \rfloor + 1)^{\text{st}}$ blocks.

To query the k^{th} block we first reduce the problem to the rank space by finding the rank of the successor of a and the predecessor of b within the block. Next, we query D_k for the non-empty subblocks within the block and use T to report the points in the subblock.

If the query to D does not return any point then either $[a,b]$ is empty or $[a,b]$ is contained fully within a block. To determine which block contains $[a,b]$ we query P to get the rank of a random point in $[a,b]$ from that we determine which block contains $[a,b]$. Afterwards we proceed within the block as described above.

Deletions. To delete a point p we first query to check that the interval $[p,p]$ is not empty. We obtain the rank k of p by querying P, and then we set the k^{th} bit in T to 1. Now we know that the point p is in the $s = (2(k \mod \lg^2 m)/\lg n)^{\text{th}}$ subblock of the $b = (k/\lg^2 m)^{\text{th}}$ block. We check if the s^{th} subblock is empty. If that is so we remove its endpoints from $D_{(k/\lg^2 m)}$. Then we check if the b^{th} block is empty. In that case we remove its endpoints from D. The expected running time is $O(\lg^{\varepsilon} m)$.

Space Analysis. P uses $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits and D contains $O(n/\lg^2 m)$ points thus uses $O(n/\lg m)$ bits. Each D_i $(1 \leq i \leq n/\lg^2 m)$ contains $O(\lg^2 m/\lg n)$ points from a universe if size $\lg^2 m$ thus uses $O(\lg^2 m \lg\lg m/\lg n)$ bits. The D_i structures use $O(n \lg\lg m/\lg n)$ bits in total. If $\lg\lg m \notin o(\lg n)$ then $n < \lg^c m$ for some constant c. In that case we use a slightly different approach. We reduce the problem to the rank space from the beginning to make the universe size n, so D uses $O(n/\lg n)$ bits and the D_i structures use $O(n \lg\lg n/\lg n)$ bits in total. The table T uses $O(\sqrt{n} \lg^3 n \lg\lg n)$ bits and finally the compressed bit vector uses $o(n)$ as long as the number of deletions is $o(n)$. In total the space remains $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits.

Construction Time and Workspace. P can be constructed in expected $O(n \lg^{\varepsilon} m)$ time using $o(n)$ extra bits of space. D can be constructed in expected $O(n/\lg^{2-\varepsilon} m)$ time using $O(1)$ extra words of space. Each D_i can be constructed in expected $O((\lg^2 m/\lg n)\lg^{\varepsilon}\lg m)$ time using $O(1)$ extra words of space, so all the D_i's can be constructed in expected $O((n/\lg n)\lg^{\varepsilon}\lg m)$ time using $O(1)$ extra words of space. T can be constructed in $o(n)$ time using $o(n)$ extra bits of space. In total the construction time and workspace are dominated by the cost of constructing P and remain the same as in Theorem 3.

▶ **Theorem 4.** *There exists a semi-dynamic succinct $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$-bit data structure that supports one-dimensional range reporting queries in $O(k+1)$ time where k is the number of points within the query, and point deletions in expected $O(\lg^{\varepsilon} m)$ time as long as the number of deletions is $o(n)$. Additionally given the point set in sorted order, this data structure can be constructed in expected $O(n\lg^{\varepsilon} m)$ time using $o(n)$-bits workspace.*

4 Fully-Dynamic Succinct One-Dimensional Point Reporting

4.1 Fully-Dynamic Structure with Amortized Updates

We first present a fully dynamic solution that uses $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$ bits of space and supports queries in $O(k)$ time and updates in amortized expected $O(\lg^{\varepsilon} m)$ time.

We divide the universe of size m into $n/\lg^2 m$ chunks of equal size and maintain a fully dynamic [14] data structure B to keep track of the nonempty chunks. B is maintained throughout the data structure updates. Whenever a point is inserted we insert both endpoints of its chunk into B. Moreover whenever a chunk becomes empty we remove its endpoints from B. For each chunk b_i $(1 \le i \le n/\lg^2 m)$ we maintain two data structures: S_i and D_i. S_i is the compressed semi-dynamic range reporting structure described in Theorem 4 and D_i is the fully dynamic data structure described in [14]. We maintain the invariant that $\text{size}(D_i) < \text{size}(S_i)/\lg^{\varepsilon} n$ for all i where $n = \sum_i \text{size}(S_i)$. Once $\text{size}(D_i) = \text{size}(S_i)/\lg^{\varepsilon} n$ we rebuild S_i and merge D_i with it. The time needed to rebuild S_i will be $O(\text{size}(S_i)\lg^{\varepsilon} m$ which we can charge to the elements inserted into D_i at a cost of $O(\lg^{2\varepsilon} m)$ per element. Moreover if the total number of elements increase by a constant factor or if $n/\lg^{\varepsilon} n$ elements were deleted from the collections S_i we rebuild the whole data structure. The time needed to rebuild the whole structure is $O(n\lg^{\varepsilon} m)$ and will be charged to the new elements inserted if the size doubles at a cost of $O(\lg^{\varepsilon} m)$ per element, or to the elements deleted at a cost of $O(\lg^{2\varepsilon} m)$ per element.

To report all the points within an interval $[a,b]$ we query B to get the non-empty chunks. Whenever a non-empty chunk i is reported we query both S_i and D_i. If $[a,b]$ is completely within one chunk we get its index $i = \lfloor b\lg^2 m/n \rfloor$, and then we query S_i and D_i.

The space used by B is at most $O(n/\lg m)$ bits. and the space used by all the D_i structures is:

$$\begin{aligned} O(n\lg(m\lg^2 m/n)/\lg^{\varepsilon} n) &= O((n\lg(m/n)/\lg^{\varepsilon} n) + (n\lg\lg n/\lg^{\varepsilon} n)) \\ &= o(\mathcal{B}(n,m)). \end{aligned}$$

The space used by all the structures S_i is $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$ bits. In total the space used is $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$ bits.

▶ **Theorem 5.** *There exist a dynamic succinct $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$-bit data structure that supports one-dimensional range reporting queries in $O(k+1)$ time where k is the number of points within the query, and updates in amortized expected $O(\lg^{\varepsilon} m)$ time.*

4.2 Fully-Dynamic Structure with Worst Case Updates

Next, we present a fully-dynamic succinct one-Dimensional range reporting structure that supports queries in $O(k)$ time and insertions and deletions in expected $O(\lg^{\varepsilon} m)$ time. Our data structure uses techniques similar to the ones presented in [9, 15, 16].

Data Structure. We define a parameter $n_f = \Theta(n)$; the value of n_f changes as n becomes too large or too small. We divide m into $(n_f/\lg^2 n_f)$ chunks each of size $((m \lg^2 n_f)/n_f)$ and we store a dynamic range reporting structure B with a universe of size $2(n_f/\lg^2 n_f)$ on the endpoints of the non-empty chunks. For each chunk b where $1 \leq b \leq (n_f/\lg^2 n_f)$ we store the following:

k_f^b an estimate of k the number of points in the chunk. $k_f^b = \Theta(k)$, the value of k_f^b changes as k becomes too large or too small.

Data Structures $\mathcal{C}_1^b, \ldots, \mathcal{C}_{\lg^{\varepsilon} n_f}^b$. These structures are the succinct semi-dynamic structures described in the previous section. They partition the chunk into sub-chunks of possibly different sizes, each containing $\Theta(k_f^b/\lg^{\varepsilon} n_f)$ points.

Data Structures $\mathcal{D}_1^b, \ldots, \mathcal{D}_{\lg^{\varepsilon} n_f}^b$. These structures are the fully dynamic structures described in [14].

$\mathcal{F}^b$ a fusion tree on the endpoints of the $\mathcal{C}_i^b$ data structures.

Queries are answered in a manner similar to the previous subsection. To report all the points within an interval $[a, b]$ we query B to get the non-empty chunks. Whenever a non-empty chunk (say the b^{th} chunk) is reported we query $\mathcal{F}^b$ to get the sub-chunks it spans. For each sub-chunk (say the s^{th} sub-chunk) we query both $\mathcal{C}_s^b$ and $\mathcal{D}_s^b$.

Insertions. To insert the new point p we compute the chunk $b = \lfloor (p \lg^2 n_f)/n_f \rfloor$ that p belongs to. If the b^{th} chunk is empty we insert its endpoints into B. Next, we check if any structure in the $\mathcal{C}^b$ collection is being rebuilt. In that case we spend $\Theta(\lg^{3\varepsilon} n_f)$ time rebuilding it. Then we determine the s^{th} sub-chunk that p belongs to using $\mathcal{F}^b$. Finally, we insert p into $\mathcal{D}_s^b$.

In each chunk we run the following background process. After each series of $\delta = k_f^b/(\lg^{2\varepsilon} n_f \lg\lg n_f)$ insertions we identify the s^{th} sub-chunk with the largest number of inserted points and rebuild $\mathcal{C}_s^b$ during the next δ updates in that chunk. The re-building works as follows. We construct a semi-dynamic data structure $\overline{\mathcal{C}}_s^b = \mathcal{C}_s^b \cup \mathcal{D}_s^b$. If a point is inserted into this sub-chunk, we store it in the additional data structure $\overline{\mathcal{D}}^b$. When $\overline{\mathcal{C}}_s^b$ is completed we set $\mathcal{C}_s^b := \overline{\mathcal{C}}_s^b$ and $\mathcal{D}_s^b := \overline{\mathcal{D}}^b$. Thus at any time only one sub-chunk of a chunk is re-built. This method guarantees that the number of inserted elements into $\mathcal{D}^b$ does not exceed $k_f^b/\lg^{\varepsilon} n$ as follows from a Theorem of Dietz and Sleator:

▶ **Lemma 6** ([3], Theorem 5). *Suppose that $x_1, \ldots, x_g$ are variables that are initially zero. Suppose that the following two steps are iterated:*

(i) *we add a non-negative real value a_i to each x_i such that $\sum a_i = 1$*

(ii) *set the largest x_i to 0.*

Then at any time $x_i \leq 1 + h_{g-1}$ for all i, $1 \leq i \leq g$, where h_i denotes the i-th harmonic number.

Let m_s be the number of inserted elements into $\mathcal{D}_s^b$ and $x_s = m_s/\delta$. Every iteration of the background process sets the largest x_s to 0 and during each iteration $\sum x_s$ increases by 1. Hence the value of x_s can be bounded from above by: $x_s \leq 1 + h_{\lg^{\varepsilon} n_f}$ for all s at all times.

Thus $m_s = O((k_f^b/\lg^{2\varepsilon} n_f \lg\lg n_f)\lg\lg n_f) = O(k_f^b/\lg^{2\varepsilon} n_f)$ for all i because $h_i = O(\lg i)$, and the total size of the $\mathcal{D}^b$ collection is $O((k_f^b/\lg^{2\varepsilon} n_f)\lg^{\varepsilon} n_f) = O(k_f^b/\lg^{\varepsilon} n_f)$.

Once the value of k_f^b becomes too big or too small we rebuild the whole chunk during the next $k_f^b/\lg^{3\varepsilon} n_f$ updates (spending $O(\lg^{4\varepsilon} n_f)$ time per update). The old chunk is locked such that only deletions are allowed. We rebuild the chunk with an updated value of k_f^b and as points are inserted into the new chunk we delete them from the old one to preserve space. If the size of the sub-chunk becomes too big we split it into two and update $\mathcal{F}^b$ accordingly.

Deletions. Deletions are similar to insertions. To delete a point p we compute the chunk $b = \lfloor (p\lg^2 n_f)/n_f \rfloor$ that p belongs to. Then we check if any structure in the $\mathcal{C}^b$ collection is being rebuilt. In that case we spend $\Theta(\lg^{3\varepsilon} n_f)$ time rebuilding it. Next, we determine the sub-chunk s that p belongs to using $\mathcal{F}^b$. Finally, we delete p from $\mathcal{C}_s^b$ and $\mathcal{D}_s^b$.

In each chunk we run a background process similar to the process run for insertions. After each series of δ deletions, we identify the s^{th} sub-chunk with the largest number of deletions and rebuild $\mathcal{C}_s^b$ during the next δ updates in that chunk. This method guarantees that the number of deleted elements in the $\mathcal{C}^b$ collection does not exceed $k_f^b/\lg^{\varepsilon} n$. If the size of a sub-chunk becomes too small we merge it with the neighboring sub-chunk and update $\mathcal{F}^b$ accordingly. Moreover if a chunk becomes empty we delete its endpoints from B.

Space Analysis. The space used by B is $O(n/\lg n)$. The space used by all the $\mathcal{C}_i$ structures in all chunks is $\mathcal{B}(n,m)+o(\mathcal{B}(n,m))$ bits. The total size of all the $\mathcal{D}$ structures is $O(n_f/\lg^{\varepsilon} n_f)$ so they use at most:

$$\begin{aligned} O(n\lg(m\lg^2 n/n)/\lg^{\varepsilon} n) &= O((n\lg(m/n)/\lg^{\varepsilon} n) + (n\lg\lg n/\lg^{\varepsilon} n)) \\ &= o(\mathcal{B}(n,m)). \end{aligned}$$

The space used by the fusion trees in all chunks is:

$$\begin{aligned} O(n\lg^{\varepsilon} n\lg(m\lg^2 n/n)/\lg^2 n) &= O((n\lg(m/n)/\lg^{2-\varepsilon} n) + (n\lg\lg n/\lg^{2-\varepsilon} n)) \\ &= o(\mathcal{B}(n,m)). \end{aligned}$$

Thus the total space is $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits.

Once the value of n_f becomes too big or too small, we rebuild the whole data structure in the background during the next $n_f/\lg^{3\varepsilon} n_f$ updates (spending $O(\lg^{4\varepsilon} n_f)$ time per update). We replace the chunks from left to right. The chunk being replaced is locked such that only deletions are allowed. We rebuild that chunk with an updated value and as points are inserted into the new chunk we delete them from the old one to preserve space.

▶ **Theorem 7.** *There exist a dynamic succinct $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$-bit data structure that supports one-dimensional range reporting queries in $O(k+1)$ time where k is the number of points within the query, and updates in expected $O(\lg^{\varepsilon} m)$ time.*

5 Succinct Static One-Dimensional Point Reporting With Fast Construction Time

In this section we prove Theorem 3. Denote by T the classic binary tree with 2^w leaves where all leaves have depth w as described in subsection 2.1. Let P be the set of nodes in T with non-empty subtrees and V the set of branching nodes in T union the leaves of T and its root. Let T_V be the tree formed from T by deleting all vertices in $T - P$ then contracting

all vertices in $P - V$. Given a node $x \in T_V$ denote by $T(x)$ its corresponding node in T, conversely, given a node $x \in V$ denote by $T_V(x)$ its corresponding node in T_V. We fix a constant $\varepsilon = 1/k$, and let $H_i = \lg^{(k-i)/k} m$ where $1 \leq i < k$. Finally, given a node u in T we define $\pi_i(u)$ to be the nearest ancestor of u whose depth is a multiple of H_i.

Data Structure. We store the coordinates of the points in $\mathcal{B}(n,m) + o(\mathcal{B}(n,m))$ bits. Also we store T_V using $4n + o(n)$ bits using the tree representation of Navarro and Sadakane [18] which allows the following operations in constant time:

lmost-leaf(i) / rmost-leaf(i): given the preorder number of a node return the preorder number of the leftmost(rightmost) leaf of node i.

leaf-rank(i): given the preorder number of a leaf i returns the number of leafs to the left of i.

In addition we store in $o(n)$ bits the index described in [6] that enables conversion between τ-names of the nodes in T_V and their preorder numbers.

To maintain the mapping between the labels of the branching nodes in T with their preorder numbers in T_V we store the following tables using Corollary 2:

M_1**:** for each node $x \in V$ with root $(T_V(x)) = T_V(x)$ we store the value $\tau_1(T_V(x))$ in a table M_1. Since T_V is a binary tree, it is possible that $T_V(x)$ belongs to two different micro trees μ_0 and μ_1. In that case we store both $p(M_0)$ and $p(M_1)$.

M_2**:** for each node $x \in V$ we store in a table M_2 the values $\tau_2(T_V(x))$, $\tau_3(T_V(x))$, and a bit that indicates to which micro tree does $T_V(x)$ belongs to if root $(T_V(x))$ belongs to two different micro trees.

M_3**:** for each node $x \in V$ we store the distance from x to $T(\text{root}\,(T_V(x)))$ in a table M_3.

Finally, given a node in P we need to compute its nearest branching ancestor. To achieve this we use the same technique as in [1] but with bootstrapping. We store $k-1$ tables $D_1, \ldots, D_{(k-1)}$ using Corollary 2. D_1 contains the distances to the nearest branching ancestor for all nodes u in P satisfying $\pi_1(u) = u$. D_i $(2 \leq i < k-1)$ contains the distances to the nearest branching ancestor for all nodes u in P satisfying the conditions $\pi_{(i-1)}(u)$ is closer to u than the nearest branching ancestor of u and $\pi_i(u) = u$. Finally, $D_{(k-1)}$ contains the distances to the nearest branching ancestor for all nodes u in P satisfying the conditions: $\pi_{(k-2)}(u)$ is closer to u than the nearest branching ancestor of u and $\pi_{(k-1)}(u) = u$, or $\pi_{(k-1)}(u)$ and $\pi_{(k-2)}(u)$ are closer to u than the nearest branching ancestor of u. More formally we define:

B_1**:** $B_1(z) = 1$ if $\pi_1(z) = z$ and $\exists u \in V$ such that $\pi_1(u) = z$, otherwise $B_1(z) = 0$.

$B_i(1 < i < k)$**:** $B_i(z) = 1$ if $B_{(i-1)}(\pi_{(i-1)}(z)) = 1$, $\pi_i(z) = z$, and $\exists u \in V$ such that $\pi_i(u) = z$, otherwise $B_i(z) = 0$

and store the following tables using Corollary 2:

D_1**:** which contain the distance to the nearest branching ancestor for all nodes u in P satisfying $\pi_1(u) = u$.

D_i $(2 \leq i < k-1)$**:** which contain the distance to the nearest branching ancestor for all nodes u in P satisfying: $B_{(i-1)}(\pi_{(i-1)}(u)) = 1$ and $\pi_i(u) = u$.

$D_{(k-1)}$**:** which contain the distance to the nearest branching ancestor for all nodes u in P satisfying: $B_{(k-2)}(\pi_{(k-2)}(u)) = 1$ and $(\pi_{(k-1)}(u) = u$ or $B_{(k-1)}(\pi_{(k-1)}(u)) = 1)$.

Query. Given a query FindAny(a, b) we first find the nearest common ancestor p of a and b. Then we get $k-1$ candidate nearest branching ancestor $v_1, \ldots, v_{(k-1)}$ of p using $D_1, \ldots, D_{(k-1)}$. Afterwards for each v_i we need to compute the preorder number of v_i in T_V. To achieve this goal we get $\tau_2(T_V(v_i)), \tau_3(T_V(v_i))$, and the bit b indicating which micro tree v_i belongs to from M_2. Next, we compute $u_i = T(\text{root}\,(T_V(v)))$ after obtaining its distance

from v_i using M_3. Afterwards we query M_1 for $\tau_1(T_V(u_i)) = p(\mu_b)$. After obtaining the τ-name of $T_V(v_i)$ we get its preorder number, and then we check the ranks of the leftmost and rightmost leaves of v_i's left and right child. If one of them is within $[a, b]$ we return its value. If for all v_i no element was found within $[a, b]$ we return that $S \cap [a, b]$ is empty.

Space Analysis. Storing the points coordinates uses $\mathcal{B}(n, m)$ bits. The tree T_V uses $4n + o(n)$ bits. The tables M_2, M_3 contain $O(n)$ entries each of size $O(\lg\lg m)$ so they use $O(n \lg\lg m)$ bits. The table M_1 contains $O(n/\lg n)$ entries each of size $O(\lg n)$ so it uses $O(n)$ bits. The table D_1 contains $O(n \lg m/\lg^{(k-1)/k} m) = O(n \lg^{\varepsilon} m)$ entries of size $O(\lg\lg m)$ bits each so it uses $O(n \lg^{\varepsilon} m \lg\lg m)$ bits. Moreover each table D_i $(1 < i < k-1)$ contains $O(n(H_{(i-1)}/H_i)) = O(n \lg^{\varepsilon} m)$ entries each of size $O(\lg\lg m)$ bits so they use a total of $O(n \lg^{\varepsilon} m \lg\lg m)$ bits. Finally, we need to bound the size of D_{k-1}. The number of entries due to $\pi_{k-1}(u) = u$ is $O(n(H_{(k-1)}/H_k)) = O(n \lg^{\varepsilon} m)$. To bound the entries due to $B_{k-1}(\pi_{k-1}(u)) = 1$ notice that the subtree T_z of height $H_{(k-1)}$ rooted at $z = \pi_{(k-1)}(u)$ will contain $s > 1$ entries, and will have at most $s + 1 < 2s$ leaves that are nodes in P. Thus it will contribute at most $(2H_{(k-1)}s)$ entries. Since there are at most $n - 1$ branching nodes the total number of entries due to $B_{(k-1)}(\pi_{(k-1)}(u)) = 1$ is $2H_{(k-1)}n = O(n \lg^{\varepsilon} m)$. D_{k-1} uses $O(n \lg^{\varepsilon} m \lg\lg m)$ bits because each entry in $D_{(k-1)}$ is of size $O(\lg\lg m)$ bits. In total the space used is $\mathcal{B}(n, m) + O(n) + O(n \lg^{\varepsilon} m \lg\lg m)$ bits.

Construction Time. In a manner similar to [1] we can identify V in $O(n)$ time, and then construct T_V also in $O(n)$ time. The tables M_1, M_2,and M_3 can be constructed in expected $O(n(\lg\lg m))$ time. Finally, the tables B_i where $1 \leq i < k$ can be constructed in expected $O(n \lg^{\varepsilon} m)$ time by identifying the $O(n \lg^{\varepsilon} m)$ entries and building the tables. The workspace is $O(n)$ words.

Reducing Space. To further reduce the space we use a well known trick and split the universe $[m]$ into n ranges $r_1, \ldots, r_n$ each of size m/n. We construct a bit vector B of size $2n$ bits with rank and select queries. B stores a zero for each range r_i followed by n_i ones where n_i is the number of points in the range r_i. To count the number of points before a range r_i we use a select query to get the position of the i^{th} zero in B, and then use a rank query to count the number of ones before that position. We store a separate data structure for each range. To locate the data structures for any range r_i within A we count the number of points in the ranges r_j for $j < i$, and then scale that number. Given a query FindAny(a, b) we check if $[a, b]$ spans a non-empty range as follows. We use a rank query to get the number of ones k before the $\lfloor (an/m) \rfloor$ zero. Then we check if the $(k+1)^{\text{th}}$ element is within $[a, b]$ and return it in that case. Otherwise we query the data structure corresponding to the $(\lceil (an/m) \rceil)^{\text{th}}$ range. The total space used is $\mathcal{B}(n, m)+O(n)+O(n(\lg (m/n))^{\varepsilon} \lg\lg (m/n)) = \mathcal{B}(n, m)+o(\mathcal{B}(n, m))+O(n)$ bits.

If $O(n)$ is not a lower order term then $n > m/c$ for some constant c. In that case we adopt a different approach and store the points in a compressed bit vector of size m. To answer a query FindAny(a, b) we use a rank query to get the number of ones k before position a, and then we use a select query to get the position of the $(k + 1)^{\text{th}}$ one. If that position is within $[a, b]$ we return it otherwise $S \cap [a, b]$ is empty. The space used is now $\mathcal{B}(n, m) + o(\mathcal{B}(n, m))$ bits.

Reducing Construction Workspace. To further improve the construction workspace we divide n into $\lg^2 m$ ranges each containing $n/\lg^2 m$ points and build a separate data structure for each of them. We note that the universe size in each range may vary. Additionally we

store a fusion tree F on the endpoints of each range. Given a query FindAny(a, b), we check if the successor of a in F is within $[a, b]$ and return it in that case. Otherwise we query the range containing the successor of a.

References

1 Stephen Alstrup, Gerth Brodal, and Theis Rauhe. Optimal static range reporting in one dimension. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 476–482. ACM, 2001.

2 Prosenjit Bose, Eric Y. Chen, Meng He, Anil Maheshwari, and Pat Morin. Succinct geometric indexes supporting point location queries. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 635–644, 2009.

3 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 365–372. ACM, 1987.

4 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct color searching in one dimension. In *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, pages 30:1–30:11, 2017.

5 Arash Farzan, J. Ian Munro, and Rajeev Raman. Succinct indices for range queries with applications to orthogonal range maxima. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming, Part I*, pages 327–338, 2012.

6 Richard F Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms (TALG)*, 2(4):510–534, 2006.

7 Mayank Goswami, Allan Grønlund Jørgensen, Kasper Green Larsen, and Rasmus Pagh. Approximate range emptiness in constant time and optimal space. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 769–775, 2015.

8 Roberto Grossi, Rajeev Raman, Satti Srinivasa Rao, and Rossano Venturini. Dynamic compressed strings with random access. In *International Colloquium on Automata, Languages, and Programming*, pages 504–515. Springer, 2013.

9 Ankur Gupta, Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. A framework for dynamizing succinct data structures. In *International Colloquium on Automata, Languages, and Programming*, pages 521–532. Springer, 2007.

10 Meng He. Succinct and implicit data structures for computational geometry. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 216–235, 2013.

11 J Ian Munro and S Srinivasa Rao. Succinct representation of data structures. In *Handbook of Data Structures and Applications*, chapter 37. Chapman and Hall/CRC, 2004.

12 Christiaan TM Jacobs and Peter Van Emde Boas. Two results on tables. *Information Processing Letters*, 22(1):43–48, 1986.

13 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 103–111. ACM, 1995.

14 Christian Worm Mortensen, Rasmus Pagh, and Mihai Patrascu. On dynamic range reporting in one dimension. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 104–111. ACM, 2005.

15 Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Dynamic data structures for document collections and graphs. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 277–289. ACM, 2015.

16 J Ian Munro and Yakov Nekrich. Compressed data structures for dynamic sequences. In *Algorithms-ESA 2015*, pages 891–902. Springer, 2015.

17 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

18 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms (TALG)*, 10(3):16, 2014.

Enumerating Vertices of 0/1-Polyhedra associated with 0/1-Totally Unimodular Matrices

Khaled Elbassioni
Masdar Institute, Khalifa University of Science and Technology, Abu Dhabi 54224, UAE
khaled.elbassioni@ku.ac.ae

Kazuhisa Makino
Research Institute for Mathematical Sciences (RIMS) Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract

We give an incremental polynomial time algorithm for enumerating the vertices of any polyhedron $P = P(A, \underline{1}) = \{x \in \mathbb{R}^n \mid Ax \geq \underline{1},\ x \geq \underline{0}\}$, when A is a totally unimodular matrix. Our algorithm is based on decomposing the hypergraph transversal problem for unimodular hypergraphs using Seymour's decomposition of totally unimodular matrices, and may be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms, Mathematics of computing → Hypergraphs

Keywords and phrases Totally unimodular matrices, Vertices of polyhedra, Vertex enumeration, Hypergraph transversals, Hypergraph decomposition, Output polynomial-time algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.18

Acknowledgements We thank Endre Boros and Vladimir Gurvich for helpful discussions.

1 Introduction

1.1 The vertex enumeration problem

The well-known Minkowski-Weyl theorem states that any convex polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ can be represented as the Minkowski sum of the convex hull of the set $\mathcal{V}(\mathcal{P})$ of its extreme points and the conic hull of the set $\mathcal{D}(\mathcal{P})$ of its extreme directions (see e.g. [29]). Given a polyhedron $\mathcal{P}$ by its linear description as the intersection of finitely many halfspaces, obtaining the set $\mathcal{V}(\mathcal{P}) \cup \mathcal{D}(\mathcal{P})$, required by the other representation, is a well-known problem, called *Vertex Enumeration* (VE) (see,. e.g.,[14, 10]), which have been extensively studied in the literature in different (but polynomially equivalent) forms, e.g., , the *facet enumeration* problem [10] or the *polytope-polyhedron problem* [25]. Clearly, the size of the extreme set $\mathcal{V}(\mathcal{P}) \cup \mathcal{D}(\mathcal{P})$ can be (and typically is) exponential in the dimension n and the number of linear inequalities m, and thus when considering the computational complexity of the vertex enumeration problem, one is usually interested in *output-sensitive* algorithms [30], i.e., those whose running time depends not only on n and m, but also on $|\mathcal{V}(\mathcal{P}) \cup \mathcal{D}(\mathcal{P})|$. Alternatively, we may consider the following, polynomially equivalent, decision variant of the problem:

Dec($\mathcal{L}; \mathcal{X} \subseteq \mathcal{C}(\mathcal{P})$): Given a polyhedron $\mathcal{P}$, represented by a system of linear inequalities $\mathcal{L}$, and a subset $\mathcal{X} \subseteq \mathcal{C}(\mathcal{P})$, is $\mathcal{X} = \mathcal{C}(\mathcal{P})$?

In this description, $\mathcal{C}(\mathcal{P})$ could be either $\mathcal{V}(\mathcal{P})$, $\mathcal{D}(\mathcal{P})$, or $\mathcal{V}(P) \cup \mathcal{D}(P)$. The problem of enumerating the elements of $\mathcal{C}(\mathcal{P})$ is said to be solvable in *incremental polynomial* time if problem Dec($\mathcal{L}; \mathcal{X} \subseteq \mathcal{C}(\mathcal{P})$) can be solved in time polynomial in the size of the description of

$\mathcal{L}$ and $\mathcal{X}$.[1] It is well-known that if the decision problem is NP-hard, then no *output* (or *total*) *polynomial-time* algorithm can generate the elements of $\mathcal{C}(\mathcal{P})$ unless P=NP (see e.g. [8]).

Vertex enumeration is an outstanding open problem in computational geometry and polyhedral combinatorics (see, e.g., [15, 25, 27]), and has numerous applications. For example, understanding the structure of the vertices helps in designing approximation algorithms for combinatorial optimization problems [33]; finding all vertices can be used for computing Nash equilibria for bimatrix games [5]. Numerous algorithmic ideas for vertex or facet enumeration have been introduced in the literature, see, e.g., [1, 2, 3, 10, 11, 15, 12, 14, 30, 28, 4].

The main result in [21] established that problem $\mathrm{Dec}(\mathcal{L}; \mathcal{X} \subseteq \mathcal{V}(\mathcal{P}))$ is NP-hard for unbounded polyhedra, more precisely, when $|\mathcal{D}(\mathcal{P})|$ is exponentially large in the input size. This negative result holds, even when restricted to 0/1-*polyhedra* [9], that is, when $\mathcal{V}(\mathcal{P}) \subseteq \{0,1\}^n$, and comes in contrast with the fact that the VE problem for 0/1-polytopes (i.e., bounded polyhedra) is known to be solvable with *polynomial delay* (that is, the vertices are generated such that the delay between any successive outputs is polynomial *only* in the *input* size) and *polynomial space* (that is, the total space used for enumerating all the vertices is polynomial in the *input* size).

1.2 VE for 0/1-Polyhedra Associated with 0/1-Totally Unimodular Matrices

Let $A \in \{0,1\}^{m \times n}$ be an $m \times n$ 0/1-matrix such that the polyhedron

$$\mathcal{P}(A, \underline{1}) = \{x \in \mathbb{R}^n \mid Ax \geq \underline{1},\ x \geq \underline{0}\} \tag{1}$$

has only integral vertices, where $\underline{1}$ (resp., $\underline{0}$) denotes the vector of all ones (resp., zeros) of appropriate dimension. Then $\mathcal{P}(A, \underline{1})$ has only n extreme directions (namely the n unit vectors in $\mathbb{R}^n$), while the vertices of P are in one-to-one correspondence with the minimal transversals of the hypergraph $\mathcal{H}[A] \subseteq 2^{[n]}$, whose characteristic vectors of hyperedges are the rows of A. One of the most important examples is when the matrix A is *totally unimodular*: in this case, the polyhedron $\mathcal{P}(A, \underline{1})$ has integral vertices, and VE is equivalent to finding all minimal transversals[2] of a *unimodular hypergraph* $\mathcal{H}[A]$. Consequently, it follows from the well-known result in [19] that all vetrices of such polyhedra can be enumerated in *quasi-polynomial* time, and hence the VE problem in this case is unlikely to be NP-hard. Polynomial time algorithms for special cases of this problem are known; for example, enumerating minimal vertex/edge covers for a bipartite graphs [16, 26], enumerating minimal hitting sets/set covers of interval hypergraphs [8], and enumerating minimal path covers/cut conjunctions in directed trees [8]. However, the complexity of the VE problem for (1) remains open, even for the totally unimodular matrices A. In this paper, we settle the complexity of the VE problem in the latter case.

▶ **Theorem 1.** *Let $A \in \{0,1\}^{m \times n}$ be a totally unimodular matrix. Then the vertices of $\mathcal{P}(A, \underline{1})$ can be enumerated in incremental polynomial time.*

---

[1] Note that if the answer to the decision problem is "NO" then a new element in $\mathcal{C}(\mathcal{P}) \setminus \mathcal{X}$ can be found by a polynomial number of calls to the decision problem.

[2] Note that, it is not possible to reduce the problem of enumerating the vertices of $\mathcal{P}(A, \underline{1})$ to that of enumerating the vertices of the 0/1 polytope $\mathcal{P}' = \{x \in \mathbb{R}^n \mid Ax \geq \underline{1},\ \underline{0} \leq x \leq \underline{1}\}$, as $\mathcal{P}'$ can have exponentially more vertcies than those of $\mathcal{P}$ (namely, the vertices of $\mathcal{P}'$ are the (not necessarily minimal) transversals of $\mathcal{H}[A]$).

A celebrated result of Seymour [31] shows that any totally unimodular matrix (with 0, ±1-entries) arises from (essentially) the so-called *network matrices*, by a small set of simple operations. Similar results for 0/1-totally unimodular matrices are derived in [32, Chapter 11], with the main building blocks replaced by 0/1-network matrices. On the other hand, it has been shown in [8] that for any polyhedron $\mathcal{P}(A, \underline{1})$, with a 0/1-network matrix A, the VE problem can be solved in incremental polynomial time. To prove Theorem 1, we show that the above mentioned decomposition of totally unimodular matrices yields a corresponding decomposition for the hypergraph transversal problem, that can be leveraged into a polynomial time algorithm for the problem. One of the natural ways to use such decomposition is to recursively partition the input polyhedron into two smaller polyhedra and then combine the outputs from the two subproblems. While such approach works for the simple cases of the decomposition (so-called 1- and 2-sum decompositions), it does not work for the more complicated case (so-called 3-sum decomposition). The main reason is that the number of vertices in either of the two subproblems may be exponentially larger than that in the original problem. To overcome this difficulty, we need to use the decomposition in a more sophisticated way, utilizing structural properties of the unimodular hypergraph $\mathcal{H}[A]$. On technical hurdle which arises is that the total input/output size of the resulting subproblems might exceed the input/output size of the original problem, which may eventually lead to an exponential blow-up in the overall running time of the algorithm in terms of the input *and* output sizes. To deal with this issue, we introduce a *volume* measure as the product of the input and output sizes, and show in each case of our decomposition that the total measure of the subproblems is smaller than the measure of the original problem.

2 Notation and Preliminaries

2.1 Hypergraphs and Transversals

Let V be a finite set. A hypergraph $\mathcal{H} \subseteq 2^V$ is a family of subsets of V. A hypergraph is called *Sperner* (*simple* or a *clutter*), if it has the property that no hyperedge contains another. For a hypergraph $\mathcal{H} \subseteq 2^V$, we denote by $\mathrm{Tr}(\mathcal{H})$ the family of *minimal transversals* of $\mathcal{H}$, i.e., (inclusion-wise) minimal subsets of V which have a nonempty intersection with each hyperedge of $\mathcal{H}$; $\mathrm{Tr}(\mathcal{H})$ is also called the *dual* of $\mathcal{H}$. Note that for the purpose of enumerating all minimal transversals of a hypergraph $\mathcal{H}$, it is enough to consider the Sperner hypergraph consisting of the minimal hyperedges of $\mathcal{H}$. We say that the hypergraph $\mathcal{H}$ is *trivial* if $\mathcal{H} = \emptyset$ or $\mathcal{H} = \{\emptyset\}$, and is *irredundant* if every $v \in V$ belongs to some $H \in \mathcal{H}$. As usual, we assume $\mathrm{Tr}(\{\emptyset\}) = \emptyset$ and $\mathrm{Tr}(\emptyset) = \{\emptyset\}$.

Given two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ with vertex set V, denote by

$$\mathcal{H}_1 \wedge \mathcal{H}_2 = \mathrm{Min}\{H_1 \cup H_2 \mid H_1 \in \mathcal{H}_1 \text{ and } H_2 \in \mathcal{H}_2\},$$
$$\mathcal{H}_1 \vee \mathcal{H}_2 = \mathrm{Min}(\mathcal{H}_1 \cup \mathcal{H}_2),$$

the *conjunction* and *disjunction* of $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively, where for hypergraph $\mathcal{H}$, $\mathrm{Min}(\mathcal{H})$ denotes the family of (inclusion-wise) minimal sets in $\mathcal{H}$. We denote by $\mathcal{H}_1 \dot{\cup} \mathcal{H}_2$ the *disjoint* union of $\mathcal{H}_1$ and $\mathcal{H}_2$. For two hypergraphs $\mathcal{H}_1 \subseteq 2^{V_1}$ and $\mathcal{H}_2 \subseteq 2^{V_2}$, we denote by $\mathcal{H}_1 \dot{\wedge} \mathcal{H}_2$ the conjunction of $\mathcal{H}_1$ and $\mathcal{H}_2$ when V_1 and V_2 are *disjoint*. By definition, $|\mathcal{H}_1 \dot{\cup} \mathcal{H}_2| = |\mathcal{H}_1| + |\mathcal{H}_2|$ and $|\mathcal{H}_1 \dot{\wedge} \mathcal{H}_2| = |\mathcal{H}_1| \cdot |\mathcal{H}_2|$.

For a hypergraph $\mathcal{H} \subseteq 2^V$ and a set $S \subseteq V$, we denote by $\mathcal{H}_S = \{H \in \mathcal{H} \mid H \subseteq S\}$ and $\mathcal{H}^S = \mathrm{Min}\{H \cap S \mid H \in \mathcal{H}\}$ the subhypergraph of $\mathcal{H}$ *induced by* S, and the *projection* of $\mathcal{H}$ on S, respectively. For $W, S \subseteq V$, we write $\mathcal{H}(W, S) = \{H \in \mathcal{H} \mid H \cap W = S\}$. Two

vertices of $\mathcal{H}$ are said to be *identical* if they belong to exactly the same hyperedges, i.e., the corresponding columns in the hyperedge-vertex incidence matrix are identical.

The following propositions are straightforward (see e.g. [6, 17, 23]).

▶ **Proposition 2.** *Given a hypergraph $\mathcal{H} \subseteq 2^V$ and a set $S \subseteq V$, the following statements hold:*

(i) $\mathrm{Tr}(\mathrm{Tr}(\mathcal{H})) = \mathrm{Min}(\mathcal{H})$,

(ii) $\mathrm{Tr}(\mathcal{H}_S) = \mathrm{Tr}(\mathcal{H})^S$ *(and hence,* $\mathrm{Tr}(\mathcal{H}^S) = \mathrm{Tr}(\mathcal{H})_S$*) and*

(iii) $|\mathrm{Tr}(\mathcal{H}_S)| \leq |\mathrm{Tr}(\mathcal{H})|$.

▶ **Proposition 3.** *Given hypergraphs $\mathcal{H}_1, \ldots, \mathcal{H}_k \subseteq 2^V$,* $\mathrm{Tr}\left(\bigvee_{i=1}^r \mathcal{H}_i\right) = \bigwedge_{i=1}^r \mathrm{Tr}(\mathcal{H}_i)$.

As a corollary of Proposition 3 we have the following.

▶ **Proposition 4.** *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph and $S_1, \ldots, S_r \subseteq V$ be subsets such that for every hyperhedge $H \in \mathcal{H}$ there exists an $i \in \{1, \ldots, r\}$ with $H \subseteq S_i$. Then* $\mathrm{Tr}(\mathcal{H}) = \bigwedge_{i=1}^r \mathrm{Tr}(\mathcal{H}_{S_i})$.

Throughout the paper, we use the notation: $n = n(\mathcal{H}) = |V|$, $m = m(\mathcal{H}) = |\mathcal{H}|$ and $k = k(\mathcal{H}) = |\mathrm{Tr}(\mathcal{H})|$.

2.2 Polyhedra

A convex polyhedron $P \subseteq \mathbb{R}^n$ is the intersection of finitely many halfspaces, determined by the *facets* of the polyhedron. A *vertex* or an *extreme point* of P is a point $v \in \mathbb{R}^n$ which cannot be represented as a convex combination of two other points of P, i.e., there exists no $\lambda \in (0,1)$ and $v_1, v_2 \in P$ such that $v = \lambda v_1 + (1-\lambda)v_2$. A *(recession) direction* of P is a vector $d \in \mathbb{R}^n$ such that $x_0 + \mu d \in P$ whenever $x_0 \in P$ and $\mu \geq 0$. An *extreme direction* of P is a direction d that cannot be written as a conic combination of two other directions, i.e., there exist no positive real numbers $\mu_1, \mu_2 \in \mathbb{R}_+$ and directions d_1, d_2 of P such that $d = \mu_1 d_1 + \mu_2 d_2$. Denote respectively by $\mathcal{V}(P)$ and $\mathcal{D}(P)$ the sets of extreme points and extreme directions of polyhedron P. A bounded polyhedron, i.e., one for which $\mathcal{D}(P) = \emptyset$ is called a *polytope*.

2.3 Totally Unimodular Matrices

A matrix $A \in \{0,1\}^{m \times n}$ is *totally unimodular* if every square subdeterminant of it has value in $\{-1, 0, 1\}$. We denote by $\mathcal{U}^{m \times n}$ the set of $m \times n$ 0/1-totally unimodular matrices. For a matrix $A \in \{0,1\}^{m \times n}$ we denote by $\mathcal{H}[A] \subseteq 2^{[n]}$ the hypergraph whose characteristic vectors of hyperedges are the rows of A. A hypergraph $\mathcal{H}$ is said to be *unimodular* [6] if $\mathcal{H} = \mathcal{H}[A]$ for a totally unimodular matrix A. Note by definition that if $\mathcal{H} \subseteq 2^V$ is unimodular then for any set $S \subseteq V$ and any subhypergraph $\mathcal{H}' \subseteq \mathcal{H}$, the hypergraph $(\mathcal{H}')^S$ is unimodular. A 0/1 matrix is said to be *ideal* (see, e.g., [13]) if the polyhedron $P = P(A, \underline{1})$ has only integral vertices. It is well-known that every totally unimodular matrix $A \in \{0,1\}^{m \times n}$ is ideal. Furthermore, the following correspondence holds.

▶ **Proposition 5** ([24])**.** *Let A be an $m \times n$ ideal matrix. Then the vertices of the polyhedron $\mathcal{P}(A, \underline{1})$ are in one-to-one correspondence with the minimal transversals of the hypergraph $\mathcal{H}[A]$.*

2.4 0/1-Network matrices

A matrix $A \in \{0,1\}^{m\times n}$ is said to be a *network* matrix if there exists a directed tree[3] T such that the rows of A one-to-one correspond to the arcs in T, and each column of A is the characteristic vector of a directed path in T. Checking if a given matrix A is a network matrix and finding the corresponding tree representation can be done in polynomial time (see e.g., [29]). We call a hypergraph $\mathcal{H}$ a *network hypergraph* if $\mathcal{H} = \mathcal{H}[A]$ for some network matrix A or its transpose. It is known that network hypergraphs can be dualized in incremental polynomial time and polynomial space:

▶ **Theorem 6** ([8]). *Let $A \in \{0,1\}^{m\times n}$ be a network matrix. Then*

(i) *all the vertices of $P(A, \underline{1})$ can be enumerated in incremental polynomial time using polynomial space;*

(ii) *all the vertices of $P(A^T, \underline{1})$ can be enumerated in incremental polynomial time using polynomial space.*

2.5 Decomposition of 0/1-totally unimodular matrices

Seymour [31] gave a decomposition theorem that allows one to decompose (in polynomial time) any 0/1-totally unimodular matrix by repeatedly applying certain operations (called *i-sums*, for $i = 1, 2, 3$) until simple building blocks are obtained; the building blocks consist of 0/1-network matrices, their transposes and a specific 5×5 0/1-matrix. For our purposes this theorem can be stated as follows.

▶ **Theorem 7** (Decomposition of unimodular hyeprgraphs). *Let $\mathcal{H} \subseteq 2^V$ be a unimodular (nontrivial) irredundant Sperner hypergraph. Then $\mathcal{H}$ is a network hypergraph, (isomorphic to) the hypergraph $\mathcal{H}_0 = \{\{1,4,5\},\{1,2,5\},\{2,3,5\},\{3,4,5\}\}$, has two identical vertices, a hyperedge consisting of a singleton, or a vertex with degree 1, or there exists a nontrivial partition $V_1 \dot\cup V_2 = V$ such that $\mathcal{H}$ can be decomposed as follows:*

- *1-sum decomposition:*
  - **(i)** *$\mathcal{H}_{V_1} \neq \emptyset$, $\mathcal{H}_{V_2} \neq \emptyset$;*
  - **(ii)** *for all $H \in \mathcal{H}$: either $H \subseteq V_1$ or $H \subseteq V_2$;*
- *2-sum decomposition: there exists a nonempty set $S \subseteq V_1$ such that*
  - **(i)** *$\mathcal{H}_{V_1} \neq \emptyset$, $\mathcal{H}(V_1, S) \neq \emptyset$, $\mathcal{H}(V_1, S)^{V_2} \neq \{\emptyset\}$;*
  - **(ii)** *for all $H \in \mathcal{H}$ with $H \cap V_1 \neq \emptyset$ and $H \cap V_2 \neq \emptyset$: $H \cap V_1 = S$;*
- *3-sum decomposition – case 1: there exist two nonempty sets $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$, such that*
  - **(i)** *$\mathcal{H}(V_1, S_1) \neq \emptyset$, $\mathcal{H}(V_1, S_1)^{V_2} \neq \{\emptyset\}$, $\mathcal{H}(V_2, S_2) \neq \emptyset$, $\mathcal{H}(V_2, S_2)^{V_1} \neq \{\emptyset\}$;*
  - **(ii)** *$|V_1| + |\mathcal{H}_{V_1} \cup \mathcal{H}(V_2, S_2)| \geq 4$, $|V_2| + |\mathcal{H}_{V_2} \cup \mathcal{H}(V_1, S_1)| \geq 4$;*
  - **(iii)** *for all $H \in \mathcal{H}$ with $H \cap V_1 \neq \emptyset$ and $H \cap V_2 \neq \emptyset$: either $H \cap V_1 = S_1$ or $H \cap V_2 = S_2$;*
- *3-sum decomposition – case 2: there exist three nonempty disjoint sets $S_0, S_1, S_2 \subseteq V_1$, such that*
  - **(i)** *$\mathcal{H}_{V_1} \neq \emptyset$, $\mathcal{H}(V_1, S_0 \cup S_1) \neq \emptyset$, $\mathcal{H}(V_1, S_0 \cup S_1)^{V_2} \neq \{\emptyset\}$, $\mathcal{H}(V_1, S_0 \cup S_2) \neq \emptyset$, $\mathcal{H}(V_1, S_0 \cup S_2)^{V_2} \neq \{\emptyset\}$;*
  - **(ii)** *for all $H \in \mathcal{H}$ with $H \cap V_1 \neq \emptyset$ and $H \cap V_2 \neq \emptyset$: either $H \cap V_1 = S_0 \cup S_1$, or $H \cap V_1 = S_0 \cup S_2$;*

[3] We say that a directed graph G is a *directed tree* if the underlying graph of G (i.e., the undirected graph obtained from G by ignoring orientation of arcs) is a tree.

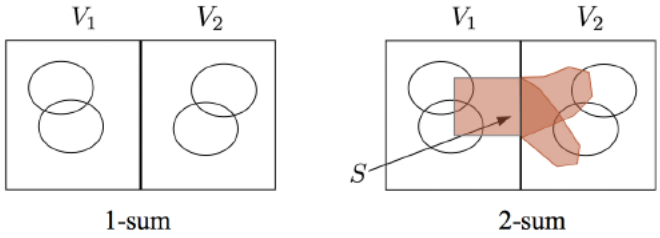


Figure 1 Decomposing a unimodular hypergraph: 1 and 2-sums.

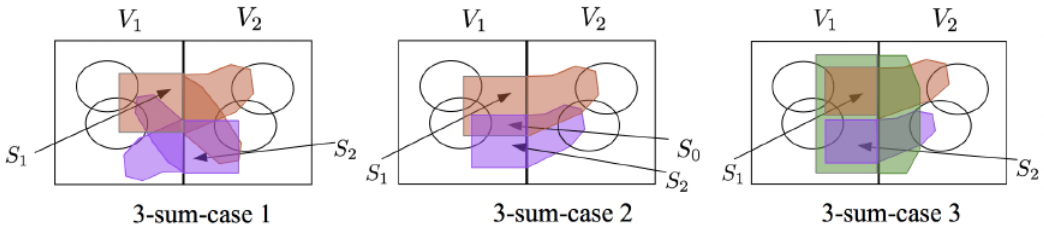


Figure 2 Decomposing a unimodular hypergraph: 3-sum.

- 3-*sum decomposition – case 3: there exist two nonempty disjoint sets* $S_1, S_2 \subseteq V_1$*, such that*
  - **(i)** $\mathcal{H}_{V_1} \neq \emptyset$ *and at least two of the following three conditions hold: (1)* $\mathcal{H}(V_1, S_1) \neq \emptyset$ *and* $\mathcal{H}(V_1, S_1)^{V_2} \neq \{\emptyset\}$*, (2)* $\mathcal{H}(V_1, S_2) \neq \emptyset$*,* $\mathcal{H}(V_1, S_2)^{V_2} \neq \{\emptyset\}$*, (3)* $\mathcal{H}(V_1, S_1 \cup S_2) \neq \emptyset$*,* $\mathcal{H}(V_1, S_1 \cup S_2)^{V_2} \neq \{\emptyset\}$*;*
  - **(ii)** $|V_1| + |\mathcal{H}_{V_1}| \geq 4$*,* $|V_2| + |\mathcal{H}_{V_2} \cup \mathcal{H}(V_1, S_1) \cup \mathcal{H}(V_1, S_2) \cup \mathcal{H}(V_1, S_1 \cup S_2)| \geq 4$*;*
  - **(iii)** *for all* $H \in \mathcal{H}$ *with* $H \cap V_1 \neq \emptyset$ *and* $H \cap V_2 \neq \emptyset$*: either* $H \cap V_1 = S_1$*,* $H \cap V_1 = S_2$*, or* $H \cap V_1 = S_1 \cup S_2$*.*

Discovering if $\mathcal{H}$ *is a network hypergraph, or isomorphic to* $\mathcal{H}_0$*, and if not finding a decomposition as above can be done in polynomial time.*

A schematic illustration of these decomposition rules is given in Figures 1 and 2.

▶ Remark. We note that the boundary condition (ii) in the 3-sum–case 1 is essential, since without insisting on this condition, any hypergraph can be decomposed according to the 3-sum–case 1 rule (take any $v \in V$ and $H \in \mathcal{H}$ such that $v \in H$ and let $V_1 = S_1 = \{v\}$, $V_2 = V \setminus \{v\}$ and $S_2 = H \setminus \{v\}$). Similarly, our analysis in the 3-sum–case 3 uses condition (ii). However, a similar condition is not needed for all other cases.

3 Decomposition of the hypergraph transversal problem

In the following, we show how to decompose the hypergraph transversal problem for a unimodular hypergraph $\mathcal{H}$, given the decomposition of $\mathcal{H}$ in Theorem 7. Such a decomposition yields naturally a recursive algorithm: each non-leaf node of the recursion tree is responsible for computing the dual of a unimodular hypergraph, while leaves involve the computation of the dual of a network hypergraph or the hypergraph $\mathcal{H}_0$. To ensure that the overall running time is polynomial, we need to bound the number of nodes of the recursion tree and the local computation time at each node, which consists of the time required for computing the decomposition and the time for combining the outputs from the recursive calls into the final output at the node. We will measure the "volume" of each subproblem to compute $\mathrm{Tr}(\mathcal{H})$

by $\mu(\mathcal{H}) = nmk = n(\mathcal{H})m(\mathcal{H})k(\mathcal{H})$. We let $T(\mu)$ be the number of nodes of the recursion subtree rooted at a node of volume μ, and let $L_1(\mu)$ and $L_2(\mu)$ be respectively the local computation time for the decomposition and combining the outputs at a node of volume μ. We stop the recursion when either $m = m(\mathcal{H})$, $n = n(\mathcal{H})$ or $k = k(\mathcal{H})$ drops below some constant C, in which case the hypergraph transversal problem can be solved in $\text{poly}(n, m)$ time using a simple procedures, such as *Berge Multiplication* [6, Chapter 2] for $n(\mathcal{H}) \leq C$ or $m(\mathcal{H}) \leq C$, and the methods in [7, 20] for $k(\mathcal{H}) \leq C$ which also show that the condition $k(\mathcal{H}) \leq C$ can be checked in $\text{poly}(n, m)$ time.

We will show by induction (on $\mu \geq 1$) that $T(\mu) \leq \mu$. We also show that $L_2(\mu) = O(\mu^c)$ for some constant $c \geq 1$. Since $L_1(\mu) = \text{poly}(n, m)$ [29, Chapter 20], it would follow then that the total time to compute $\text{Tr}(\mathcal{H})$ is at most $O(\mu^{1+c}) + \text{poly}(\mu)$, which is polynomial in n, m, and k. This would give a *total* polynomial-time algorithm for computing $\text{Tr}(\mathcal{H})$ which can be converted into an *incremental* polynomial-time algorithm by standard methods [22, 8]. Thus, we shall assume in the sequel that n, m, k are larger than any desired constant C.

Without loss of generality we assume that the input hypergraph is Sperner and irredundant, and this assumption is maintained for all hypergraphs arising as inputs to the recursive subproblems. We may also assume that $\mathcal{H}$ has neither singleton hyperedge nor vertex of degree 1 (i.e., contained in exactly one hyperedge). Indeed, if $\mathcal{H}$ contains a singleton hyperedge $H = \{v\}$, then by the Sperner property, no other hyperedge of $\mathcal{H}$ contains v. In this case, and also in the case when $\mathcal{H}$ has a vertex v contained exactly in one hyperedge $H \in \mathcal{H}$, $\text{Tr}(\mathcal{H})$ can be computed as follows:

$$\text{Tr}(\mathcal{H}) = \text{Tr}((\mathcal{H} \setminus \{H\}) \dot\cup \{H\}) = \text{Tr}((\mathcal{H} \setminus \{H\}) \wedge \text{Tr}(\{H\}), \tag{2}$$

where $\text{Tr}(\{H\} = \{\{w\} \mid w \in H\}$. By Proposition 2 (iii), $|\text{Tr}((\mathcal{H} \setminus \{H\})| \leq k(\mathcal{H})$ and thus, $\mu(\mathcal{H}') \leq (n-1)(m-1)k \leq \mu(\mathcal{H}) - 1$, where $\mathcal{H}' := \mathcal{H} \setminus \{H\}$ is the the subhypergraph induced by $V \setminus \{v\}$. Thus, we get by induction that $T(\mu(\mathcal{H})) \leq 1 + T(\mu(\mathcal{H}')) \leq \mu(\mathcal{H})$. Moreover, by (2), $\text{Tr}(\mathcal{H})$ can be computed from $\text{Tr}((\mathcal{H} \setminus \{H\})$ in $\text{poly}(n, m, k)$ time.

Finally, we may also assume that $\mathcal{H}$ does not have two identical vertices. Indeed, if it has two such vertices v, v' then we can reduce the problem by calling the algorithm on the hypergraph $\mathcal{H}' = \{H \setminus \{v'\} \mid H \in \mathcal{H}\}$ instead of $\mathcal{H}$. Then the dual of $\mathcal{H}$ can be obtained as follows:

$$\text{Tr}(\mathcal{H}) = \text{Tr}(\mathcal{H}') \dot\cup \{(T \setminus \{v\} \cup \{v'\}) \mid T \in \text{Tr}(\mathcal{H}'),\ v \in T\}. \tag{3}$$

Note that (3) implies that $k(\mathcal{H}') \leq k(\mathcal{H})$ and hence $\mu(\mathcal{H}') \leq (n-1)mk \leq \mu(\mathcal{H}) - 1$. Thus, in this case, we get the recurrence $T(\mu) \leq 1 + T(\mu - 1)$, which gives by induction on $\mu \geq 1$ that $T(\mu) \leq 1 + (\mu - 1) \leq \mu$. Moreover, by (3), $\text{Tr}(\mathcal{H})$ can be computed from $\text{Tr}(\mathcal{H}')$ in $\text{poly}(n, m, k)$ time.

The 1-sum decomposition case is straightforward and is omitted. As a warm-up, we present the 2-sum case in Section 3.1, and then move to the more complicated 3-sum decomposition. In fact, by Theorem 7 the latter can be divided into 3 subcases: we present 3-sum-case 1 in Section 3.2. Then for each of the 3-sum cases 2 and 3, we can identify 4 subcases. Due to space limitations, we present only the simplest cases (3-sum case 2-I and 3-sum case 2-II) in Section 3.3 and refer the reader to [18] for the remaining cases.

We will use the following simple facts in our analysis of the running time of the algorithm.

▶ **Fact 8.** *Let α, β, N, M be positive integers such that $\alpha \leq N/2$ and $\beta \leq M/2$. Consider the maximization problem:*

$$\begin{aligned} z^* = \max \quad & x_1y_1 + x_2y_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq N, \\ & y_1 + y_2 \leq M, \\ & x_1, x_2 \geq \alpha, \\ & y_1, y_2 \geq \beta, \\ & x_1, x_2, y_1, y_2 \in \mathbb{Z}. \end{aligned}$$

Then $z^* = \alpha\beta + (N - \alpha)(M - \beta)$.

Proof. Let $(x_1^*, x_2^*, y_1^*, y_2^*)$ be an optimal solution. Clearly, $x_2^* = N - x_1^*$ and $y_2^* = M - y_1^*$. Without loss of generality assume that $x_1^* \geq \frac{N}{2}$. If $y_1^* < M - \beta$, then $(x_1^*, N - x_1^*, y_1^* + 1, M - y_1^* - 1)$ is also an optimal solution since

$$\begin{aligned} x_1^*(y_1^* + 1) + (N - x_1^*)(M - (y_1^* + 1)) &= x_1^*y_1^* + (N - x_1^*)(M - y_1^*) + 2x_1^* - N \\ &\geq x_1^*y_1^* + (N - x_1^*)(M - y_1^*). \end{aligned}$$

Thus we conclude in this case that $(x_1^*, N - x_1^*, M - \beta, \beta)$ is also an optimal solution. A symmetric argument shows that $(N-\alpha, \alpha, M-\beta, \beta)$ is an optimal solution of the maximization problem. ◀

▶ **Fact 9.** *Let x_i, y_i, for $i = 1, \ldots, h$, and M be positive integers such that $\sum_{i=1}^h x_iy_i \leq M$. Then $\sum_{i=1}^h (x_i + y_i) \leq M + h$.*

Proof. For $i = 1, \ldots, h$, let $\alpha_i = x_iy_i$. Note that the function $f(x_i) = x_i + \frac{\alpha}{x_i}$ is convex in $x_i > 0$, and hence $\max\{x_i + y_i \mid x_iy_i = \alpha_i,\ x_i \geq 1, y_i \geq 1\}$ is achieved at the boundary $(x_i, y_i) = (1, \alpha_i)$ or $(x_i, y_i) = (\alpha_i, 1)$. The claim follows by summing the inequality $x_i + y_i \leq \alpha_i + 1$ over $i = 1, \ldots, h$. ◀

3.1 2-sum decomposition

Given a nontrivial partition $V_1 \dot\cup V_2 = V$ and a nonempty set $S \subseteq V_1$ such that for all $H \in \mathcal{H}$ with $H \cap V_1 \neq \emptyset$ and $H \cap V_2 \neq \emptyset$: $H \cap V_1 = S$, we have the following decomposition of the dual hypergraph by Proposition 3:

$$\operatorname{Tr}(\mathcal{H}) = \operatorname{Tr}(\mathcal{H}_{V_1} \dot\cup \mathcal{H}_{V_2 \cup S}) = \operatorname{Tr}(\mathcal{H}_{V_1}) \wedge \operatorname{Tr}(\mathcal{H}_{V_2 \cup S}), \tag{4}$$

as $\mathcal{H} = \mathcal{H}_{V_1} \dot\cup \mathcal{H}_{V_2 \cup S}$ (note that $\mathcal{H}_{V_1} \cap \mathcal{H}_{V_2 \cup S} = \emptyset$ since $\mathcal{H}$ is Sperner and that both $\mathcal{H}_{V_1}$ and $\mathcal{H}_{V_2 \cup S}$ are unimodular). Thus in this case we get the recurrence:

$$T(\mu) \leq 1 + T(\mu_1) + T(\mu_2), \tag{5}$$

where $\mu = \mu(\mathcal{H})$, $\mu_1 = \mu(\mathcal{H}_{V_1})$, and $\mu_2 = \mu(\mathcal{H}_{V_2 \cup S})$. Let $n_1 = n(\mathcal{H}_{V_1}) = |V_1|$, $m_1 = |\mathcal{H}_{V_1}|$, $k_1 = |\operatorname{Tr}(\mathcal{H}_{V_1})|$, $n_2 = n(\mathcal{H}_{V_2 \cup S}) = |V_2| + |S|$, $m_2 = |\mathcal{H}_{V_2 \cup S}|$, and $k_2 = |\operatorname{Tr}(\mathcal{H}_{V_2 \cup S})|$. Note that $n_1, n_2, m_1, m_2 \geq 1$ by the assumptions of the 2-sum case (Theorem 7) and hence $\mu_1, \mu_2 \geq 1$. Then

$$\mu_1 + \mu_2 = n_1m_1k_1 + n_2m_2k_2 \leq (n-1)(m_1 + m_2)k = (n-1)mk \leq \mu - 1, \tag{6}$$

where $k_1 \leq k$ and $k_2 \leq k$ by Proposition 2 (iii). It follows by induction from (5) that

$$T(\mu) \leq 1 + \mu_1 + \mu_2 \leq 1 + (\mu - 1) = \mu. \tag{7}$$

Note that $\mathrm{Tr}(\mathcal{H})$ can be computed from $\mathrm{Tr}(\mathcal{H}_{V_1})$ and $\mathrm{Tr}(\mathcal{H}_{V_2 \cup S})$ using (4) in time $L_2(\mu) = \mathrm{poly}(n, m, k)$.

3.2 3-sum decomposition - case 1

Assume we are given a nontrivial partition $V_1 \dot\cup V_2 = V$ and two nonempty sets $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$, such that for all $H \in \mathcal{H}$ with $H \cap V_1 \neq \emptyset$ and $H \cap V_2 \neq \emptyset$: either $H \cap V_1 = S_1$ or $H \cap V_2 = S_2$. Let $n_1 = |V_1|$, $n_2 = |V_2|$, $m_1 = |\mathcal{H}_{V_1} \cup \mathcal{H}(V_2, S_2)|$ and $m_2 = |\mathcal{H}_{V_2} \cup \mathcal{H}(V_1, S_1)|$. It is also assumed in this case that $n_1, n_2, m_1, m_2 \geq 1$, $n_1 + n_2 = n$, $m_1 + m_2 = m$, $n_1 + m_1 \geq 4$ and $n_2 + m_2 \geq 4$.

We consider two cases:

Case I: there is no hyperedge $H \in \mathcal{H}$ such that $H \subseteq S_1 \cup S_2$. Note that this, together with assumption (i) of the 3-sum–case 1 in Theorem 7, implies that $S_1 \subset V_1$ and $S_2 \subset V_2$. In this case, we have the following decomposition of the dual hypergraph:

$$\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\mathcal{H}_{V_1 \cup S_2} \dot\cup \mathcal{H}_{V_2 \cup S_1}) = \mathrm{Tr}(\mathcal{H}_{V_1 \cup S_2}) \wedge \mathrm{Tr}(\mathcal{H}_{V_2 \cup S_1}), \tag{8}$$

(Note by assumption that $\mathcal{H}_{V_1 \cup S_2} \cap \mathcal{H}_{V_2 \cup S_1} = \emptyset$.) Thus in this case we get the recurrence:

$$T(\mu) \leq 1 + T(\mu_1) + T(\mu_2), \tag{9}$$

where $\mu = \mu(\mathcal{H})$, $\mu_1 = \mu(\mathcal{H}_{V_1 \cup S_2})$, and $\mu_2 = \mu(\mathcal{H}_{V_2 \cup S_1})$. Let $n_1' = n(\mathcal{H}_{V_1 \cup S_2}) = |V_1| + |S_2|$, $m_1 = |\mathcal{H}_{V_1 \cup S_2}|$, $k_1 = |\mathrm{Tr}(\mathcal{H}_{V_1 \cup S_2})|$, $n_2' = n(\mathcal{H}_{V_2 \cup S_1}) = |V_2| + |S_1|$, $m_2 = |\mathcal{H}_{V_2 \cup S_1}|$, and $k_2 = |\mathrm{Tr}(\mathcal{H}_{V_2 \cup S_1})|$. Then

$$\mu_1 + \mu_2 = n_1' m_1 k_1 + n_2' m_2 k_2 \leq (n-1)(m_1 + m_2)k \leq \mu - 1, \tag{10}$$

where $k_1 \leq k$ and $k_2 \leq k$ by Proposition 2 (iii). It follows by induction from (9) that

$$T(\mu) \leq 1 + \mu_1 + \mu_2 \leq \mu. \tag{11}$$

Note that $\mathrm{Tr}(\mathcal{H})$ can be computed from $\mathrm{Tr}(\mathcal{H}_{V_1 \cup S_2})$ and $\mathrm{Tr}(\mathcal{H}_{V_2 \cup S_1})$ using (8) in time $L_2(\mu) = \mathrm{poly}(n, m, k)$.

Case II: there is a hyperedge $H_0 \in \mathcal{H}$ such that $H_0 \subseteq S_1 \cup S_2$. Note that $H_0 \cap S_1 \neq \emptyset$ and $H_0 \cap S_2 \neq \emptyset$ since otherwise by the simplicity of $\mathcal{H}$ we are in the 2-sum case. Without loss of generality, assume that $H_0 \cap V_1 = S_1$ and $H_0 \cap V_2 \subseteq S_2$. We assume that $\mathcal{H}(V_1, S_1)$ and $\mathcal{H}(V_2, S_2)$ are not empty; otherwise, we are in the 1-sum or 2-sum case. Given these assumptions, we use the following decomposition of the dual hypergraph:

$$\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\mathcal{H}_1 \cup \mathcal{H}_2) = \mathrm{Tr}(\mathcal{H}_1) \wedge \mathrm{Tr}(\mathcal{H}_2), \tag{12}$$

where $\mathcal{H}_1 = \mathcal{H}_{V_1} \cup \mathcal{H}(V_2, S_2) \cup \{H_0\}$ and $\mathcal{H}_2 = \mathcal{H}_{V_2} \dot\cup \mathcal{H}(V_1, S_1)$. Note that H_0 is the the only hyperedge that belongs to both $\mathcal{H}_1$ and $\mathcal{H}_2$. Note also that neither $\mathcal{H}_1$ nor $\mathcal{H}_2$ may be an induced subhypergraph of $\mathcal{H}$ (i.e., the form $\mathcal{H}_S$ for some $S \subseteq V$) since there are hyperedges $H \subseteq S_1 \cup S_2$ that may not be included in $\mathcal{H}_1$ and $\mathcal{H}_2$. Hence, Proposition 2 cannot be used to bound the sizes of $\mathrm{Tr}(\mathcal{H}_1)$ and $\mathrm{Tr}(\mathcal{H}_2)$. Nevertheless, due to the special structure of the

decomposition in this case, we can use the bounds given in Lemma 11 below instead. Let $\bar{\mathcal{H}}_1 \subseteq 2^{V_1 \cup \{v_2\}}$ (resp., $\bar{\mathcal{H}}_2 \subseteq 2^{V_2 \cup \{v_1\}}$) be the hypergraph obtained from $\mathcal{H}_1$ (resp., $\mathcal{H}_2$) by replacing S_2 (resp., S_1) by a *new* single vertex v_2 (resp., v_1), that is,

$$\bar{\mathcal{H}}_1 = \mathcal{H}_{V_1} \cup \bar{\mathcal{H}}(V_2, S_2) \cup \{\bar{H}_0\}, \qquad \bar{\mathcal{H}}_2 = \mathcal{H}_{V_2} \cup \bar{\mathcal{H}}(V_1, S_1),$$

where $\bar{\mathcal{H}}(V_2, S_2) = \{(H \setminus S_2) \cup \{v_2\} \mid H \in \mathcal{H}(V_2, S_2)\}$, $\bar{H}_0 = (H_0 \setminus S_2) \cup \{v_2\}$, and $\bar{\mathcal{H}}(V_1, S_1) = \{(H \setminus S_1) \cup \{v_1\} \mid H \in \mathcal{H}(V_1, S_1)\}$.

▶ **Lemma 10.** *If $\mathcal{H}$ is unimodular, then both $\bar{\mathcal{H}}_1$ and $\bar{\mathcal{H}}_2$ are unimodular.*

Proof. Let v be an arbitrary vertex in $H_0 \cap S_2$. Then the (hyperedge-vertex) incidence matrix of the hypergraph $\bar{\mathcal{H}}_1$ is a submatrix of that of $\mathcal{H}$, with rows restricted to $\mathcal{H}_{V_1} \cup \mathcal{H}(V_2, S_2) \cup \{H_0\}$, and columns restricted to $V_1 \cup \{v\}$. This shows that this submatrix is totally unimodular. A similar argument shows that $\bar{\mathcal{H}}_2$ is also unimodular. ◀

▶ **Lemma 11.** $|\operatorname{Tr}(\bar{\mathcal{H}}_1)| \leq |\operatorname{Tr}(\mathcal{H})|$ *and* $|\operatorname{Tr}(\bar{\mathcal{H}}_2)| \leq |\operatorname{Tr}(\mathcal{H})|$.

Proof. We prove the claim that $|\operatorname{Tr}(\bar{\mathcal{H}}_1)| \leq |\operatorname{Tr}(\mathcal{H})|$; the other claim can be proved similarly.

It is enough to show that for every minimal transversal $T \in \operatorname{Tr}(\bar{\mathcal{H}}_1)$, there is a minimal transversal $T' \in \operatorname{Tr}(\mathcal{H})$ such that for any distinct $T_1, T_2 \in \operatorname{Tr}(\bar{\mathcal{H}}_1)$, T_1' and T_2' are distinct.

Let $\mathcal{T}_1 = \{T \in \operatorname{Tr}(\bar{\mathcal{H}}_1) : v_2 \notin T\}$ and $\mathcal{T}_2 = \operatorname{Tr}(\bar{\mathcal{H}}_1) \setminus \mathcal{T}_1$. Consider first $T \in \mathcal{T}_1$. By assumption $T \cap S_1 \neq \emptyset$ since T has a nonempty intersection with $\bar{H}_0$. It follows that the only hyperedges of $\mathcal{H}$ having empty intersection with T are those in $\mathcal{H}_{V_2}$. Note that none of these hyperedges are contained in S_2 since $\mathcal{H}$ is Sperner. This implies that $\mathcal{H}_{V_2}^{V_2 \setminus S_2} \neq \{\emptyset\}$ and therefore $\operatorname{Tr}(\mathcal{H}_{V_2}^{V_2 \setminus S_2}) \neq \emptyset$. Let T'' be an arbitrary minimal transversal in $\operatorname{Tr}(\mathcal{H}_{V_2}^{V_2 \setminus S_2})$. Then it is easy to see that $T' = T \cup T''$ is in $\operatorname{Tr}(\mathcal{H})$.

Consider now $T \in \mathcal{T}_2$. By the minimality of T, there is a hyperedge $H \in \bar{\mathcal{H}}(V_2, S_2) \cup \{\bar{H}_0\}$ such that $H \cap T = \{v_2\}$. Furthermore, for every $v \in T \setminus \{v_2\}$, there is an $H \in \mathcal{H}_{V_1}$ such that $T \cap H = \{v\}$. Let $\mathcal{H}(T) = \{H \in \mathcal{H} \mid H \cap T \setminus \{v_2\} = \emptyset\}$ and note that $\mathcal{H}(T)^{V_2}$ is nontrivial. Pick $T'' \in \operatorname{Tr}(\mathcal{H}(T)^{V_2})$ arbitrarily. Then it is easy to see that $T' = (T \setminus \{v_2\}) \cup T''$ is in $\operatorname{Tr}(\mathcal{H})$.

Finally, note that for any distinct $T_1, T_2 \in \mathcal{T}_1$ (resp., $T_1, T_2 \in \mathcal{T}_2$), the constructed minimal transversals $T_1', T_2' \in T(\mathcal{H})$ are distinct. Moreover, for $T_1 \in \mathcal{T}_1$ and $T_2 \in \mathcal{T}_2$, T_1' and T_2' are distinct because $T_1' \cap S_2 = \emptyset$ while $T_2' \cap S_2 \neq \emptyset$. ◀

To compute (12), we find $\operatorname{Tr}(\bar{\mathcal{H}}_1)$ and $\operatorname{Tr}(\bar{\mathcal{H}}_2)$ recursively. Then $\operatorname{Tr}(\mathcal{H}_1)$ and $\operatorname{Tr}(\mathcal{H}_2)$ are given by the following claim.

▶ **Lemma 12.** *Let $\mathcal{T}_1 = \{T \in \operatorname{Tr}(\bar{\mathcal{H}}_1) \mid v_2 \notin T\}$, $\mathcal{T}_2 = \{T \in \operatorname{Tr}(\bar{\mathcal{H}}_1) \mid v_2 \in T,\ S_1 \cap T = \emptyset\}$, $\mathcal{T}_3 = \operatorname{Tr}(\bar{\mathcal{H}}_1) \setminus (\mathcal{T}_1 \cup \mathcal{T}_2)$, $\mathcal{T}_1' = \{T \in \operatorname{Tr}(\bar{\mathcal{H}}_2) \mid v_1 \notin T\}$ and $\mathcal{T}_2' = \operatorname{Tr}(\bar{\mathcal{H}}_2) \setminus \mathcal{T}_1'$. Then*

$$\operatorname{Tr}(\mathcal{H}_1) = \mathcal{T}_1 \,\dot\cup\, \{(T \setminus \{v_2\}) \cup \{v\} \mid v \in H_0 \cap S_2 \text{ and } T \in \mathcal{T}_2\} \,\dot\cup\, \{(T \setminus \{v_2\}) \cup \{v\} \mid v \in S_2 \text{ and } T \in \mathcal{T}_3\}, \tag{13}$$

$$\operatorname{Tr}(\mathcal{H}_2) = \mathcal{T}_1' \,\dot\cup\, \{(T \setminus \{v_1\}) \cup \{v\} \mid v \in S_1 \text{ and } T \in \mathcal{T}_2'\}. \tag{14}$$

Proof. Let us prove (13), since the proof of (14) is similar. Suppose $T \in \operatorname{Tr}(\mathcal{H}_1)$. If $T \cap S_2 = \emptyset$ then (it is easy to see that) $T \in \mathcal{T}_1$. If $T \cap S_2 \neq \emptyset$ then by minimality of T, $|T \cap S_2| = 1$; let $T \cap S_2 = \{v\}$. If $T \cap S_1 = \emptyset$ then necessarily $v \in H_0$, in which case $(T \setminus \{v\}) \cup \{v_2\} \in \mathcal{T}_2$; otherwise v can be any element in S_2, and hence, $(T \setminus \{v\}) \cup \{v_2\} \in \mathcal{T}_3$. On the other direction, if $T \in \mathcal{T}_1$ then clearly $T \in \operatorname{Tr}(\mathcal{H}_1)$; if $T \in \mathcal{T}_2$ then $T \cap \bar{H}_0 = \{v_2\}$

which implies that $(T \setminus \{v_2\}) \cup \{v\} \in \mathrm{Tr}(\mathcal{H}_1)$ for every $v \in H_0 \cap S_2$; finally, if $T \in \mathcal{T}_3$ then there is a hyperedge $H \in \bar{\mathcal{H}}(V_2, S_2)$ such that $H \cap T = \{v_2\}$, which implies in turn that $(T \setminus \{v_2\}) \cup \{v\} \in \mathrm{Tr}(\mathcal{H}_1)$. ◀

Note that $\mathrm{Tr}(\mathcal{H})$ can be computed from $\mathrm{Tr}(\bar{\mathcal{H}}_1)$ and $\mathrm{Tr}(\bar{\mathcal{H}}_2)$ using (12) and Lemma 12 in time $L_2(\mu) = \mathrm{poly}(n, m, k)$. Now, we bound $T(\mu)$.

Let $n_1' = n(\bar{\mathcal{H}}_1) = n_1 + 1$, $m_1' = |\bar{\mathcal{H}}_1| \in \{m_1, m_1 + 1\}$, $k_1 = |\mathrm{Tr}(\bar{\mathcal{H}}_1)|$, $n_2' = n(\bar{\mathcal{H}}_2) = n_2 + 1$, $m_2 = |\bar{\mathcal{H}}_2|$, and $k_2 = |\mathrm{Tr}(\bar{\mathcal{H}}_2)|$. By the decomposition, $n_1' + n_2' = n + 2$ and $m_1' + m_2 = m + 1$, and by Lemma 11, $k_1 \le k$ and $k_2 \le k$. Note that $n_1', n_2' \ge 2$, $m_1', m_2 \ge 1$, $n_1 + m_1 \ge 4$, and $n_2' m_2 \ge 4$, by the assumptions of the 3-sum case in Theorem 7.

We consider 3 subcases.

Case II-I: $2 \le n_1' \le 3$. Then a simple procedure will be used to compute $\mathrm{Tr}(\bar{\mathcal{H}}_1)$, and hence we need only to recurse on $\bar{\mathcal{H}}_2$, giving the simpler recurrence: $T(\mu) \le 2 + T(\mu_2)$. Note that $m_2 \le m - 2$ since $n_1 \le 2$ implies $m_1 \ge 2$ and hence $m_2 = m - m_1 \le m - 2$. Since $\mu_2 = n_2' m_2 k_2 \le n(m-2)k \le \mu - 2$, we get by induction that

$$T(\mu) \le 2 + \mu_2 \le \mu. \tag{15}$$

Case II-II: $n_2' = 2$. Then a simple procedure will be used to compute $\mathrm{Tr}(\bar{\mathcal{H}}_2)$, and hence we need only to recurse on $\bar{\mathcal{H}}_1$, giving the simpler recurrence: $T(\mu) \le 2 + T(\mu_1)$. As above, $m_1 \le m - 3$ implying that $\mu_1 = n_1' m_1' k_2 \le n(m-2)k \le \mu - 2$, and giving by induction again that $T(\mu) \le \mu$.

Case II-III: $n_1' \ge 4$ and $n_2' \ge 3$. We first note that $m_1', m_2 \ge 2$. Indeed, if $m_1' = 1$ (resp., $m_2 = 1$), then $\mathcal{H}_{V_1} = \emptyset$ and $\mathcal{H}(V_2, S_2) = \{H_0\}$ (resp., $\mathcal{H}_{V_2} = \emptyset$ and $\mathcal{H}(V_1, S_1) = \{H_0\}$). Since we assume that $\mathcal{H}$ does not have identical vertices, we must have $n_1 = 1$ (resp., $n_2 = 1$). In either case we get a contradiction to the boundary assumtpions (ii) of the 3-sum–case 1 in Theorem 7. Lemmas 10 and 12 imply that, in this case, we get the recurrence:

$$T(\mu) \le 1 + T(\mu_1) + T(\mu_2), \tag{16}$$

where $\mu = \mu(\mathcal{H})$, $\mu_1 = \mu(\bar{\mathcal{H}}_1)$, and $\mu_2 = \mu(\bar{\mathcal{H}}_2)$.

Then by Fact 8, applied with $x_1 = n_1'$, $y_1 = m_1'$, $x_2 = n_2'$, $y_2 = m_2$, $N = n + 2$, $M = m + 1$, $\alpha = 3$ and $\beta = 2$, we get (as $n \ge 5$ and $m \ge 3$)

$$\begin{aligned} \mu_1 + \mu_2 &= n_1' m_1' k_1 + n_2' m_2 k_2 \le (n_1' m_1' + n_2' m_2)k \le ((n-1)(m-1) + 6)k \\ &= nmk - (n + m - 7)k \le \mu - 1. \end{aligned} \tag{17}$$

It follows by induction from (16) that

$$T(\mu) \le 1 + \mu_1 + \mu_2 \le \mu. \tag{18}$$

3.3 3-sum decomposition - case 2

Let $\mathcal{H}_1 = \mathcal{H}_{V_1}$ and $\mathcal{H}_2 = \mathcal{H}_{V_2}$. By Theorem 7, we have three nonempty disjoint sets S_0, S_1, S_2 in V_2, and the following two families are nonempty:

$$\mathcal{F}_1 = \{H \in \mathcal{H} \mid H \cap V_1 = S_0 \cup S_2, H \cap V_2 \ne \emptyset\}, \tag{19}$$

$$\mathcal{F}_2 = \{H \in \mathcal{H} \mid H \cap V_1 = S_0 \cup S_1, H \cap V_2 \ne \emptyset\}. \tag{20}$$

Note that $V_1, V_2 \neq \emptyset$, $\mathcal{H}_1 \neq \emptyset$, and $\mathcal{H}$ can be partitioned in the following way.

$$\mathcal{H} = \mathcal{H}_1 \dot\cup \mathcal{H}_2 \dot\cup \mathcal{F}_1 \dot\cup \mathcal{F}_2, \tag{21}$$

where $\dot\cup$ denotes the disjoint union. For $i = 0, 1, 2$, let

$$\mathcal{T}_i = \{T \in \mathrm{Tr}(\mathcal{H}_1) \mid T \cap S_i \neq \emptyset, T \cap S_j = \emptyset \ (j \neq i)\}, \tag{22}$$

and let

$$\mathcal{T} = \mathrm{Tr}(\mathcal{H}_1) \setminus (\mathcal{T}_0 \cup \mathcal{T}_1 \cup \mathcal{T}_2). \tag{23}$$

By definition, we have

$$\mathrm{Tr}(\mathcal{H}_1) = \mathcal{T} \dot\cup \mathcal{T}_0 \dot\cup \mathcal{T}_1 \dot\cup \mathcal{T}_2. \tag{24}$$

Let

$$\mathcal{P} = \mathcal{H}_{S_0 \cup S_1 \cup S_2} \ (= \{H \in \mathcal{H} \mid H \subseteq S_0 \cup S_1 \cup S_2\}). \tag{25}$$

We separatly consider the following 4 cases.

Case I: $\mathcal{P} = \emptyset$.
Case II: $\mathcal{P} = \{S_0 \cup S_1 \cup S_2\}$.
Case III: $\mathcal{P} \neq \emptyset, \{S_0 \cup S_1 \cup S_2\}$ and $\mathcal{T} \neq \emptyset$.
Case IV: $\mathcal{P} \neq \emptyset, \{S_0 \cup S_1 \cup S_2\}$ and $\mathcal{T} = \emptyset$.

Case I

$\mathcal{H}$ can be partitioned into $\mathcal{H}_1$ and $\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$, i.e., $\mathcal{H} = \mathcal{H}_1 \dot\cup \mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$ and $\mathcal{H}_1$, $\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2} \neq \emptyset$. Since $\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\mathcal{H}_1) \wedge \mathrm{Tr}(\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2})$, we obtain $\mathrm{Tr}(\mathcal{H})$ by computing $\mathrm{Tr}(\mathcal{H}_1)$ and $\mathrm{Tr}(\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2})$. Let $n_1 = |V_1|$, $m_1 = |\mathcal{H}_1|$, $k_1 = |\mathrm{Tr}(\mathcal{H}_1)|$, $n_2' = |S_0 \cup S_1 \cup S_2 \cup V_2|$, $m_2' = |\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}|$, $k_2 = |\mathrm{Tr}(\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2})|$. Similar to the 2-sum decomposition case, we can show that $T(\mu) \leq \mu$ and the computation of $\mathrm{Tr}(\mathcal{H})$ can be done in time $L_2(\mu) = \mathrm{poly}(n, m, k)$.

Case II

We consider two cases: (1) $|\mathcal{H}_1| \geq 2$ and (2) $|\mathcal{H}_1| = 1$.

(1) $|\mathcal{H}_1| \geq 2$. Let $\mathcal{G}$ be a hypergraph obtained from $\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$ by replacing S_0, S_1, and S_2 by new vertices v_0, v_1 and v_2, respectively. For any hyperedge $H \in \mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$, $H \cap S_i \neq \emptyset$ implies that $S_i \subseteq H$. Thus $\mathcal{G}$ is well-defined. Note that $\mathrm{Tr}(\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2})$ can be obtained from $\mathrm{Tr}(\mathcal{G})$ in polynomial time by replacing v_i with any element in S_i. Since $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$, we have $\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\mathcal{H}_1) \wedge \mathrm{Tr}(\mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2})$. We thus decompose $\mathcal{H}$ into $\mathcal{H}_1$ and $\mathcal{G}$. Namely we compute $\mathrm{Tr}(\mathcal{H})$ from $\mathrm{Tr}(\mathcal{H}_1)$ and $\mathrm{Tr}(\mathcal{G})$. Since $|\mathrm{Tr}(\mathcal{H}_1)|, |\mathrm{Tr}(\mathcal{G})| \leq |\mathrm{Tr}(\mathcal{H})| \ (= k)$, this can be done in time $L_2(\mu) = \mathrm{poly}(n, m, k)$.

Let us next show that $T(\mu) \leq \mu$. Let $n_1 = |V_1|$, $m_1 = |\mathcal{H}_1|$, $k_1 = |\mathrm{Tr}(\mathcal{H}_1)|$, $n_2' = |V_2| + 3$, $m_2' = |\mathcal{G}|$, and $k_2 = |\mathrm{Tr}(\mathcal{G})|$. Note that $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2}$, $\mathcal{H}_1 \cap \mathcal{H}_{S_0 \cup S_1 \cup S_2 \cup V_2} = \{S_0 \cup S_1 \cup S_2\}$. This, together with definition and the discussion above, implies that

$$1 \leq n_1, n_2' \leq n, \ n_1 + n_2' = n + 3, \ 2 \leq m_1, m_2' \leq m - 1, \ m_1 + m_2' = m + 1, \ k_1, k_2 \leq k. \tag{26}$$

Thus we have

$$n_1 m_1 k_1 + n_2' m_2' k_2 \leq (n_1 m_1 + n_2' m_2')k \leq (n(m-1)+6)k \leq nmk - 1 \tag{27}$$

where Fact 8 is used for the second inequality, and the third ineuqality is obtained by assuming that n is at least 7. It follows from (27) that $T(\mu) \leq \mu$. We recall that $\mathrm{Tr}(\mathcal{H})$ is directly computed from $\mathcal{H}$ if at least one of n, m, and k is bounded by some constant C. Thus in this case we have $T(\mu) = 1$, which also satisfies $T(\mu) \leq \mu$.

(2) $|\mathcal{H}_1| = 1$. In this case, we have $H_1 = \{S_0 \cup S_1 \cup S_2\}$. Therefore, the following lemma is satisfied.

▶ **Lemma 13.** *Let $\mathcal{H}$ be a hypergraph that satisfies (21) and $\mathcal{H}_1 = \{S_0 \cup S_1 \cup S_2\}$. Then we have* $\mathrm{Tr}(\mathcal{H}) = \{\{v\} \mid v \in S_0\} \dot{\wedge} \mathrm{Tr}(\mathcal{H}_2) \dot{\cup} \mathrm{Tr}(H^{V \setminus S_0})$.

Proof. From the definition, it is not difficult to see that $\mathrm{Tr}(\mathcal{H}) \supseteq \{\{v\} \mid v \in S_0\} \dot{\wedge} \mathrm{Tr}(\mathcal{H}_2) \dot{\cup} \mathrm{Tr}(H^{V \setminus S_0})$.

For the converse inclusion, let $T \in \mathrm{Tr}(\mathcal{H})$. If $T \cap S_0 = \emptyset$, then T is contained in $\mathrm{Tr}(H^{V \setminus S_0})$. Assume next that $T \cap S_0 \neq \emptyset$. For any $i = 0, 1, 2$ and any hyperedge $H \in \mathcal{H}$, $H \cap S_i \neq \emptyset$ inplies that $S_0 \subseteq H$. This means that $|T \cap S_0| = 1$ and $T \cap S_i = \emptyset$ for $i = 1, 2$. Moreover, we have $T \cap V_2 \in \mathrm{Tr}(\mathcal{H}_2)$, which completes the converse inclusion. ◀

Note that $\mathcal{H}_2, \mathcal{H}^{V \setminus S_0} \neq \{\emptyset\}$, and hence $\mathrm{Tr}(\mathcal{H}_2), \mathrm{Tr}(H^{V \setminus S_0}) \neq \emptyset$. Based on Lemma 13, we decompose $\mathcal{H}$ into $\mathcal{H}_2$ and $\mathcal{H}^{V \setminus S_0}$. Namely, we compute $\mathrm{Tr}(\mathcal{H})$ from $\mathrm{Tr}(\mathcal{H}_2)$ and $\mathrm{Tr}(H^{V \setminus S_0})$ in time $L_2(\mu) = \mathrm{poly}(n, m, k)$.

Let $n_1' = |V_2|$, $m_1' = |\mathcal{H}_2|$, $k_1' = |\mathrm{Tr}(\mathcal{H}_2)|$, $n_2' = n - |S_0|$, $m_2' = |\mathcal{H}^{V \setminus S_0}|$, and $k_2' = |\mathrm{Tr}(\mathcal{H}^{V \setminus S_0})|$. Then we have $n_1', n_2' \leq n-1$, $m_1', m_2' \leq m$, and $k_1', k_2' \leq k-1$ and $k_1' + k_2' \leq k$. Thus we have $n_1' m_1' k_1' + n_2' m_2' k_2' \leq (n-1)(m-1)k \leq nmk - 1$, where the last inequality is obtained from $n \geq 3$. This implies that $T(\mu) \leq \mu$.

References

1. S. D. Abdullahi. *Vertex Enumeration and Counting for Certain Classes of Polyhedra*. PhD thesis, Computing (Computer Algorithms) Leeds University U.K., 2003.
2. D. Avis, B. Bremner, and R. Seidel. How good are convex hull algorithms. *Computational Geometry: Theory and Applications*, 7:265–302, 1997.
3. D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8(3):295–313, 1992.
4. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.
5. D. Avis, G. D. Rosenberg, R. Savani, and B. von Stengel. Enumeration of Nash equilibria for two-player games. *Economic Theory*, 42(1):9–37, 2010.
6. C. Berge. *Hypergraphs*. Elsevier-North Holland, Amsterdam, 1989.
7. J. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive boolean functions. *Information and Computation*, 123(1):50–63, 1995.
8. E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. Generating vertices of polyhedra and related problems of monotone generation. In D. Avis, D. Bremner, and A. Deza, editors, *Proceedings of the Centre de Recherches Mathématiques at the Université de Montréal, special issue on Polyhedral Computation*, volume 49, 2009.
9. E. Boros, K. Elbassioni, V. Gurvich, and H. R. Tiwary. The negative cycles polyhedron and hardness of checking some polyhedral properties. *Annals OR*, 188(1):63–76, 2011.

10 D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete and Computational Geometry*, 20:333–357, 1998.

11 M. R. Bussieck and M. E. Lübbecke. The vertex set of a 0/1 polytope is strongly $\mathcal{P}$-enumerable. *Computational Geometry: Theory and Applications*, 11(2):103–109, 1998.

12 V. Chvátal. *Linear Programming*. Freeman, San Francisco, CA, 1983.

13 G. Cornuéjols. *Combinatorial Optimization: Packing and Covering*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 2001.

14 M. E. Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8:381–402, 1983.

15 M. E. Dyer and L. G. Proll. An algorithms for determining all extreme points of a convex polytope. *Mathematical Programming*, 12:81–96, 1977.

16 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

17 T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

18 K. Elbassioni and K. Makino. Enumerating vertices of 0/1-polyhedra associated with 0/1-totally unimodular matrices. *CoRR*, abs/1707.03914, 2017. `arXiv:1707.03914`.

19 M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.

20 V. Gurvich and L. Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Applied Mathematics*, 96-97(1):363–373, 1999.

21 L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. *Discrete & Computational Geometry*, 39(1-3):174–190, 2008.

22 L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

23 E. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9:558–565, 1980.

24 A. Lehman. On the width-length inequality, mimeographic notes. *Mathematical Programming*, 17:403–417, 1979.

25 L. Lovász. Combinatorial optimization: some problems and trends. DIMACS Technical Report 92-53, Rutgers University, 1992.

26 N. Mishra and L. Pitt. Generating all maximal independent sets of bounded-degree hypergraphs. In *COLT '97: Proceedings of the 10th annual conference on Computational learning theory*, pages 211–217, 1997.

27 M. E. Pfetsch. *The Maximum Feasible Subsystem Problem and Vertex-Facet Incidences of Polyhedra*. Dissertation, TU Berlin, 2002.

28 J.S. Provan. Efficient enumeration of the vertices of polyhedra associated with network **LP**'s. *Mathematical Programming*, 63(1):47–64, 1994.

29 A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

30 R. Seidel. *Output-size sensitive algorithms for constructive problems in computational geometry*. Computer science, Cornell University, Ithaka, NY, 1986.

31 P. D. Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory, Series B*, 28(3):305–359, 1980.

32 K. Truemper. *Matroid Decomposition*. Academic Press, 1992.

33 V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

The Parameterized Hardness of the k-Center Problem in Transportation Networks

Andreas Emil Feldmann[1]
Department of Applied Mathematics, Charles University, Prague, Czechia
feldmann.a.e@gmail.com

Dániel Marx[2]
Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
dmarx@cs.bme.hu

Abstract

In this paper we study the hardness of the k-Center problem on inputs that model transportation networks. For the problem, an edge-weighted graph $G = (V, E)$ and an integer k are given and a center set $C \subseteq V$ needs to be chosen such that $|C| \leq k$. The aim is to minimize the maximum distance of any vertex in the graph to the closest center. This problem arises in many applications of logistics, and thus it is natural to consider inputs that model transportation networks. Such inputs are often assumed to be planar graphs, low doubling metrics, or bounded highway dimension graphs. For each of these models, parameterized approximation algorithms have been shown to exist. We complement these results by proving that the k-Center problem is W[1]-hard on planar graphs of constant doubling dimension, where the parameter is the combination of the number of centers k, the highway dimension h, and even the treewidth t. Moreover, under the Exponential Time Hypothesis there is no $f(k, t, h) \cdot n^{o(t+\sqrt{k+h})}$ time algorithm for any computable function f. Thus it is unlikely that the optimum solution to k-Center can be found efficiently, even when assuming that the input graph abides to all of the above models for transportation networks at once!

Additionally we give a simple parameterized $(1 + \varepsilon)$-approximation algorithm for inputs of doubling dimension d with runtime $(k^k/\varepsilon^{O(kd)}) \cdot n^{O(1)}$. This generalizes a previous result, which considered inputs in D-dimensional L_q metrics.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Theory of computation → Facility location and clustering, Theory of computation → Problems, reductions and completeness

Keywords and phrases k-center, parameterized complexity, planar graphs, doubling dimension, highway dimension, treewidth

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.19

Related Version `https://arxiv.org/abs/1802.08563`

1 Introduction

Given a graph $G = (V, E)$ with positive edge lengths $\ell : E \to \mathbb{Q}^+$, the k-Center problem asks to find k center vertices such that every vertex of the graph is as close as possible to one of centers. More precisely, if $\text{dist}(u, v)$ denotes the length of the shortest path between u

[1] Supported by project CE-ITI (GAČR no. P202/12/G061) of the Czech Science Foundation.
[2] Supported by ERC Consolidator Grant SYSTEMATICGRAPH (No. 725978)

and v according to the edge lengths ℓ, let $B_v(\rho) = \{u \in V \mid \text{dist}(u, v) \leq \rho\}$ be the *ball of radius ρ around v*. A solution to k-Center is a set $C \subseteq V$ of *centers* such that $|C| \leq k$, and the objective is to minimize the *cost* of the solution, which is the smallest value ρ for which $\bigcup_{v \in C} B_v(\rho) = V$. This problem has numerous applications in logistics where easily accessible locations need to be chosen on a map under a budget constraint. For instance, a budget may be available to build k hospitals, shopping malls, or warehouses. These should be placed so that the distance from each point on the map to the closest facility is minimized.

The k-Center problem is NP-hard [28], and so *approximation algorithms* [28, 29] as well as *parameterized algorithms* [8, 11] have been developed for this problem. The former are algorithms that use polynomial time to compute an *α-approximation*, i.e., a solution that is at most α times worse than the optimum. For the latter, a *parameter* p is given as part of the input, and an optimum solution is computed in $f(p) \cdot n^{O(1)}$ time for some computable function f independent of the input size n. The rationale behind such an algorithm is that it solves the problem efficiently in applications where the parameter is small. If such an algorithm exists for a problem it is called *fixed-parameter tractable (FPT)* for p. Another option is to consider *parameterized approximation algorithms* [21, 22], which compute an α-approximation in $f(p) \cdot n^{O(1)}$ time for some parameter p.

By a result of Hochbaum and Shmoys [18], on general input graphs, a polynomial time 2-approximation algorithm exists, and this approximation factor is also best possible, unless P=NP. A natural parameter for k-Center is the number of centers k, for which however the problem is W[2]-hard [9]. In fact it is even W[2]-hard [16] to compute a $(2 - \varepsilon)$-approximation for any $\varepsilon > 0$, and thus parametrizing by k does not help to overcome the polynomial-time inapproximability. For structural parameters such as the vertex-cover number or the feedback-vertex-set number the problem remains W[1]-hard [19], even when combining with the parameter k. For each of the more general structural parameters treewidth and cliquewidth, an *efficient parameterized approximation scheme (EPAS)* was shown to exist [19], i.e., a $(1 + \varepsilon)$-approximation can be computed in $f(\varepsilon, w) \cdot n^{O(1)}$ time for any $\varepsilon > 0$, if w is either the treewidth or the cliquewidth, and n is the number of vertices.

Arguably however, graphs with low treewidth or cliquewidth do not model transportation networks well, since grid-like structures with large treewidth and cliquewidth can occur in road maps of big cities. As we focus on applications for k-Center in logistics, here we consider more natural models for transportation networks. These include *planar graphs*, *low doubling metrics* such as the Euclidean or Manhattan plane, or the more recently studied *low highway dimension graphs*. Our main result is that k-Center is W[1]-hard on all of these graph classes *combined*, even if adding k and the treewidth as parameters. Before introducing these graph classes, let us formally state our theorem.

▶ **Theorem 1.** *Even on weighted planar graphs of doubling dimension $O(1)$, the k-Center problem is W[1]-hard for the combined parameter (k, t, h), where t is the treewidth and h the highway dimension of the input graph. Moreover, under ETH there is no $f(k, t, h) \cdot n^{o(t+\sqrt{k+h})}$ time algorithm[3] for the same restriction on the input graphs, for any computable function f.*

A *planar graph* can be drawn in the plane without crossing edges. Such graphs constitute a realistic model for road networks, since overpasses and tunnels are relatively rare. It is known [25] that also for planar graphs no $(2 - \varepsilon)$-approximation can be computed in polynomial time, unless P=NP. On the positive side, k-Center is FPT [9] on unweighted planar graphs for the combined parameter k and ρ. However, typically if k is small then ρ is

[3] Here $o(t + \sqrt{k+h})$ means any function $g(t + \sqrt{k+h})$ such that $g(x) \in o(x)$.

large and vice versa, and thus the applications for this combined parameter are rather limited. If the parameter is only k, then an $n^{O(\sqrt{k})}$ XP-algorithm exists for planar graphs [24]. By a very recent result [20] the k-Center problem on weighted planar graphs admits an *efficient polynomial-time bicriteria approximation scheme*, which for any $\varepsilon > 0$ in $f(\varepsilon) \cdot n^{O(1)}$ time computes a solution that uses at most $(1+\varepsilon)k$ centers and approximates the optimum with at most k centers within a factor of $1+\varepsilon$. This algorithm implies an EPAS for parameter k on weighted planar graphs, since setting $\varepsilon = \min\{\varepsilon', \frac{1}{2k}\}$ forces the algorithm to compute a $(1+\varepsilon')$-approximation in time $f(k, \varepsilon') \cdot n^{O(1)}$ using at most $(1+\varepsilon)k \leq k + \frac{1}{2}$ centers, i.e., at most k centers as k is an integer. This observation is complemented by our hardness result by showing that it is necessary to approximate the solution when parametrizing by k in weighted planar graphs.

▶ **Definition 2.** The *doubling dimension* of a metric (X, dist) is the smallest $d \in \mathbb{R}$ such that for any $r > 0$, every ball of radius $2r$ is contained in the union of at most 2^d balls of radius r. The doubling dimension of a graph is the doubling dimension of its shortest-path metric.

Since a transportation network is embedded on a large sphere (namely the earth), a reasonable model is to assume that the shortest-path metric abides to the Euclidean L_2-norm. In cities, where blocks of buildings form a grid of streets, it is reasonable to assume that the distances are given by the Manhattan L_1-norm. Every metric for which the distance function is given by the L_q-norm in D-dimensional space $\mathbb{R}^D$ has doubling dimension $O(D)$. Thus a road map, which is embedded into $\mathbb{R}^2$ can reasonably be assumed to have constant doubling dimension. It is known [23] that k-Center is W[1]-hard for parameter k in two-dimensional Manhattan metrics. Also, no polynomial time $(2-\varepsilon)$-approximation algorithm exists for k-Center in two-dimensional Manhattan metrics [13], and no $(1.822-\varepsilon)$-approximation for two-dimensional Euclidean metrics [13]. On the positive side, Agarwal and Procopiuc [4] showed that for any L_q metric in D dimensions, the k-Center problem can be solved optimally in time $n^{O(k^{1-1/D})}$, and an EPAS exists for the combined parameter (ε, k, D). We generalize the latter to any metric of doubling dimension d, as formalized by the following theorem.

▶ **Theorem 3.** *Given a metric of doubling dimension d, a $(1+\varepsilon)$-approximation for k-Center can be computed in $(k^k/\varepsilon^{O(kd)}) \cdot n^{O(1)}$ time.*

Theorem 1 complements this result by showing that it is necessary to approximate the cost of the solution if parametrizing by k and d.

▶ **Definition 4.** The *highway dimension* of a graph G is the smallest $h \in \mathbb{N}$ such that, for some universal constant $c \geq 4$, for every $r \in \mathbb{R}^+$ and every ball $B_{cr}(v)$ of radius cr, there is a set $H \subseteq B_{cr}(v)$ of *hubs* such that $|H| \leq h$ and every shortest path of length more than r lying in $B_{cr}(v)$ contains a hub of H.

The highway dimension was introduced by Abraham et al. [3] as a formalization of the empirical observation by Bast et al. [5, 6] that in a road network, starting from any point A and travelling to a sufficiently far point B along the quickest route, one is bound to pass through some member of a sparse set of "access points", i.e., the hubs. In contrast to planar and low doubling graphs, the highway dimension has the potential to model not only road networks, but more general transportation networks such as those given by air-traffic or public transportation. This is because in such networks longer connections tend to be serviced through larger and sparser stations, which act as hubs. Abraham et al. [3] were able to prove that certain shortest-path heuristics are provably faster in low highway dimension

graphs than in general graphs. They specifically chose the constant $c = 4$ in their original definition, but later work by Feldmann et al. [14] showed that when choosing any constant $c > 4$ in the definition, the structure of the resulting graphs can be exploited to obtain quasi-polynomial time approximation schemes for problems such as Travelling Salesman or Facility Location. Note that increasing the constant c in Definition 4 restricts the class of graphs further. Other definitions of the highway dimension exist as well [1, 2, 3, 14] (see [15, Section 9] for a detailed discussion).

Later, Becker et al. [7] used the framework introduced by Feldmann et al. [14] to show that whenever $c > 4$ there is an EPAS for k-Center parameterized by ε, k, and h. Note that the highway dimension is always upper bounded by the vertex-cover number, as every edge of any non-trivial path is incident to a vertex cover. Hence the aforementioned W[1]-hardness result by Katsikarelis et al. [19] for the combined parameter k and the vertex-cover number proves that it is necessary to approximate the optimum when using k and h as the combined parameter. When parametrizing only by the highway dimension but not k, it is not even known if a *parameterized approximation scheme (PAS)* exists, i.e., an $f(\varepsilon, h) \cdot n^{g(\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for some functions f, g. However, under the *Exponential Time Hypothesis (ETH)* [8], by [16] there is no algorithm with doubly exponential $2^{2^{o(\sqrt{h})}} \cdot n^{O(1)}$ runtime to compute a $(2-\varepsilon)$-approximation for any $\varepsilon > 0$. The same paper [16] also presents a 3/2-approximation for k-Center with runtime $2^{O(kh \log h)} \cdot n^{O(1)}$ for a more general definition of the highway dimension than the one given in Definition 4. In contrast to the result of Becker et al. [7], it is not known whether a PAS exists when combining this more general definition of h with k as a parameter. Theorem 1 complements these results by showing that even of planar graphs of constant doubling dimension, for the combined parameter (k, h) no FPT algorithm exists, unless FPT=W[1]. Therefore approximating the optimum is necessary, regardless of whether h is according to Definition 4 or the more general one from [16], and regardless of how restrictive Definition 4 is made by increasing the constant c.

▶ **Definition 5.** A *tree decomposition* of a graph $G = (V, E)$ is a tree T each of whose nodes v is labelled by a bag $X_v \subseteq V$ of vertices of G, and has the following properties:

(a) $\bigcup_{v \in V(T)} X_v = V$,

(b) for every edge $\{u, w\} \in E$ there is a node $v \in V(T)$ such that X_v contains both u and w,

(c) for every $v \in V$ the set $\{u \in V(T) \mid v \in X_u\}$ induces a connected subtree of T.

The *width* of the tree decomposition is $\max\{|X_v| - 1 \mid v \in V(T)\}$. The *treewidth* t of a graph G is the minimum width among all tree decompositions for G.

As mentioned above, arguably, bounded treewidth graphs are not a good model for transportation networks. Also it is already known that k-Center is W[1]-hard for this parameter, even when combining it with k [19]. We include this well-studied parameter here nonetheless, since the reduction of our hardness result in Theorem 1 implies that k-Center is W[1]-hard even for weighted planar graphs when *combining* any of the parameters k, h, d, and t. As noted in [15], these parameters are not bounded in terms of each other, i.e., they are incomparable. Furthermore, the doubling dimension is in fact bounded by a constant in Theorem 1. Hence, even if one were to combine all the models presented above and assume that a transportation network is planar, embeddable into some constant dimensional L_q metric, has bounded highway dimension, and even has bounded treewidth, it is unlikely that the k-Center problem can be solved efficiently. Thus it seems unavoidable to approximate the problem in transportation networks, when developing fast algorithms.

1.1 Related work

The k-Center problem is a fairly general clustering problem and therefore finds further applications in, for instance, image processing and data-compression. The above mentioned very recent efficient bicriteria approximation scheme [20] improves on a previous (non-efficient) bicriteria approximation scheme [12], which for any $\varepsilon > 0$ and weighted planar input graph computes a $(1+\varepsilon)$-approximation with at most $(1+\varepsilon)k$ centers in time $n^{f(\varepsilon)}$ for some function f (note that in contrast to above, such an algorithm does not imply a PAS for parameter k). The paper by Demaine et al. [9] on the k-Center problem in unweighted planar graphs also considers the so-called class of *map graphs*, which is a superclass of planar graphs that is not minor-closed. They show that the problem is FPT on unweighted map graphs for the combined parameter (k, ρ). Also for the tree-depth, k-Center is FPT [19]. A closely related problem to k-Center is the ρ-Dominating Set problem, in which ρ is given and the number k of centers covering a given graph with k balls of radius ρ needs to be minimized. As this generalizes the Dominating Set problem, no $(\ln(n) - \varepsilon)$-approximation is possible in polynomial time [10], unless P=NP, and computing an $f(k)$-approximation is W[1]-hard [27] when parametrizing by k, for any computable function f.

2 The reduction

In this section we give a reduction from the Grid Tiling with Inequality ($\mathrm{GT}_{\leq}$) problem, which is defined as follows. Given κ^2 non-empty sets $S_{i,j} \subseteq [n]^2$ of pairs of integers, where $i, j \in [\kappa]$, the task is to select one pair $s_{i,j} \in S_{i,j}$ for each set such that

- if $s_{i,j} = (a, b)$ and $s_{i+1,j} = (a', b')$ then $a \leq a'$, whenever $i \leq \kappa - 1$, and
- if $s_{i,j} = (a, b)$ and $s_{i,j+1} = (a', b')$ then $b \leq b'$, whenever $j \leq \kappa - 1$.

The $\mathrm{GT}_{\leq}$ problem is W[1]-hard [8] for parameter κ, and moreover, under ETH has no $f(\kappa) \cdot n^{o(\kappa)}$ time algorithm for any computable function f.

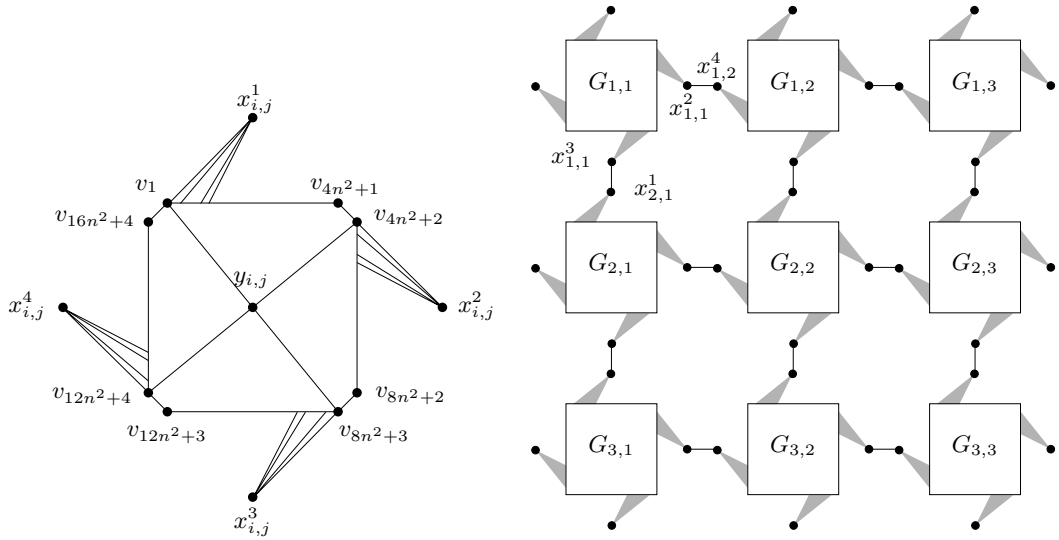
Construction

Given an instance $\mathcal{I}$ of $\mathrm{GT}_{\leq}$ with κ^2 sets, we construct the following graph $G_{\mathcal{I}}$. First, for each set $S_{i,j}$, where $1 \leq i, j \leq \kappa$, we fix an arbitrary order on its elements, so that $S_{i,j} = \{s_1, \ldots, s_\sigma\}$, where $\sigma \leq n^2$. We then construct a gadget $G_{i,j}$ for $S_{i,j}$, which contains a cycle $O_{i,j}$ of length $16n^2 + 4$ for which each edge has length 1 (see Figure 1(a)). Additionally we introduce five vertices $x^1_{i,j}$, $x^2_{i,j}$, $x^3_{i,j}$, $x^4_{i,j}$, and $y_{i,j}$. If $O_{i,j} = (v_1, v_2, \ldots, v_{16n^2+4}, v_1)$ then we connect these five vertices to the cycle as follows. The vertex $y_{i,j}$ is adjacent to the four vertices v_1, v_{4n^2+2}, v_{8n^2+3}, and v_{12n^2+4}, with edges of length $2n^2 + 1$ each. For every $\tau \in [\sigma]$ and $s_\tau \in S_{i,j}$, if $s_\tau = (a, b)$ we add the four edges

- $x^1_{i,j} v_\tau$ of length $\ell'_a = 2n^2 - \frac{a}{n+1}$,
- $x^2_{i,j} v_{\tau+4n^2+1}$ of length $\ell_b = 2n^2 + \frac{b}{n+1} - 1$,
- $x^3_{i,j} v_{\tau+8n^2+2}$ of length $\ell_a = 2n^2 + \frac{a}{n+1} - 1$, and
- $x^4_{i,j} v_{\tau+12n^2+3}$ of length $\ell'_b = 2n^2 - \frac{b}{n+1}$.

We say that the element s_τ corresponds to the four vertices v_τ, $v_{\tau+4n^2+1}$, $v_{\tau+8n^2+2}$, and $v_{\tau+12n^2+3}$. Note that s_1 (which always exists) corresponds to the four vertices adjacent to $y_{i,j}$. Note also that $2n^2 - 1 < \ell_a, \ell'_a, \ell_b, \ell'_b < 2n^2$, since $a, b \in [n]$.

The gadgets $G_{i,j}$ are now connected to each other in a grid-like fashion (see Figure 1(b)). That is, for $i \leq \kappa - 1$ we introduce a path between $x^3_{i,j}$ and $x^1_{i+1,j}$ that has $n+2$ edges, each of length $\frac{1}{n+2}$. Analogously, for $j \leq \kappa - 1$ we add a path between $x^2_{i,j}$ and $x^4_{i,j+1}$ with $n+2$ edges of length $\frac{1}{n+2}$ each. Note that these paths all have length 1.



(a) The gadget $G_{i,j}$ in the reduction.

(b) Connecting the gadgets in a grid-like fashion.

Figure 1 The reduction.

The resulting graph $G_{\mathcal{I}}$ forms an instance of k-Center with $k = 5\kappa^2$. We claim that the instance $\mathcal{I}$ of $\mathrm{GT}_{\leq}$ has a solution if and only if the optimum solution to k-Center on $G_{\mathcal{I}}$ has cost at most $2n^2$. We note at this point that the reduction would still work when removing the vertices $y_{i,j}$ and decreasing k to $4\kappa^2$. However their existence will greatly simplify analysing the doubling dimension of $G_{\mathcal{I}}$ in Section 3.

A solution to a $\mathrm{GT}_{\leq}$ instance implies cost at most $2n^2$ for the k-Center instance

Recall that we fixed an order of each set $S_{i,j}$, so that each element $s_\tau \in S_{i,j}$ corresponds to four equidistant vertices on cycle $O_{i,j}$ with distance $4n^2 + 1$ between consecutive such vertices on the cycle. If $s_\tau \in S_{i,j}$ is in the solution to the $\mathrm{GT}_{\leq}$ instance $\mathcal{I}$, let $C_{i,j} = \{v_\tau, v_{\tau+4n^2+1}, v_{\tau+8n^2+2}, v_{\tau+12n^2+3}, y_{i,j}\}$ contain the vertices of $O_{i,j}$ corresponding to s_τ in addition to $y_{i,j}$. The solution to the k-Center instance $G_{\mathcal{I}}$ is given by the union $\bigcup_{i,j\in[\kappa]} C_{i,j}$, which are exactly $5\kappa^2$ centers in total.

Let us denote the set containing the four vertices of $C_{i,j} \cap V(O_{i,j})$ by $C^O_{i,j}$, and note that each of these four vertices covers $4n^2 + 1$ vertices of $O_{i,j}$ with balls of radius $2n^2$, as each edge of $O_{i,j}$ has length 1. Since the distance between any pair of centers in $C^O_{i,j}$ is at least $4n^2 + 1$, these four sets of covered vertices are pairwise disjoint. Thus the total number of vertices covered by $C^O_{i,j}$ on $O_{i,j}$ is $16n^2 + 4$, i.e. all vertices of the cycle $O_{i,j}$ are covered. Recall that for the lengths of the edges between the vertices $x^1_{i,j}$, $x^2_{i,j}$, $x^3_{i,j}$, and $x^4_{i,j}$ and the cycle $O_{i,j}$ we have $\ell_a, \ell'_a, \ell_b, \ell'_b < 2n^2$. Hence the centers in $C^O_{i,j}$ also cover $x^1_{i,j}$, $x^2_{i,j}$, $x^3_{i,j}$, and $x^4_{i,j}$ by balls of radius $2n^2$.

Now consider a path connecting two neighbouring gadgets, e.g., let P be the path connecting $x^2_{i,j}$ and $x^4_{i,j+1}$. The center sets $C^O_{i,j}$ and $C^O_{i,j+1}$ contain vertices corresponding to the respective elements $s \in S_{i,j}$ and $s' \in S_{i,j+1}$ of the solution to the $\mathrm{GT}_{\leq}$ instance. This means that if $s = (a, b)$ and $s' = (a', b')$ then $b \leq b'$. Thus the closest centers of $C^O_{i,j}$ and

$C^O_{i,j+1}$ are at distance $\ell_b + 1 + \ell'_{b'}$ from each other, as P has length 1. From $b \leq b'$ we get

$$\ell_b + 1 + \ell'_{b'} = 2n^2 + \frac{b}{n+1} - 1 + 1 + 2n^2 - \frac{b'}{n+1} \leq 4n^2.$$

Therefore all vertices of P are covered by the balls of radius $2n^2$ around the two closest centers of $C^O_{i,j}$ and $C^O_{i,j+1}$. Analogously, we can also conclude that any path connecting some vertices $x^1_{i,j}$ and $x^3_{i+1,j}$ are covered, using the fact that if $(a,b) \in S_{i,j}$ and $(a',b') \in S_{i+1,j}$ are in the solution to the $\mathrm{GT}_{\leq}$ instance then $a \leq a'$.

Finally, the remaining center vertices in $\bigcup_{i,j\in[\kappa]} C_{i,j} \setminus C^O_{i,j}$ cover the additional vertex $y_{i,j}$ in each gadget $G_{i,j}$.

A k-Center instance with cost at most $2n^2$ implies a solution to the $\mathrm{GT}_{\leq}$ instance

We first prove that in any solution to the k-Center instance $G_{\mathcal{I}}$ of cost at most $2n^2$, each cycle $O_{i,j}$ must contain exactly four centers. Recall that $\ell_a, \ell'_a, \ell_b, \ell'_b > 2n^2 - 1$, that $y_{i,j}$ is incident to four edges of length $2n^2 + 1$ each, and that each edge of $O_{i,j}$ has length 1. Now consider the vertices v_{4n^2+1}, v_{8n^2+2}, v_{12n^2+3}, and v_{16n^2+4}, each of which is not connected by an edge to any vertex $x^h_{i,j}$, where $h \in [4]$, nor to $y_{i,j}$. Thus each of these four vertices must be covered by centers on the cycle $O_{i,j}$ if the radius of each ball is at most $2n^2$. Furthermore, the distance between each pair of these four vertices is at least $4n^2 + 1$, which means that any solution of cost at most $2n^2$ needs at least four centers on $O_{i,j}$ to cover these four vertices.

Each vertex $y_{i,j}$ must be contained in any solution of cost at most $2n^2$, since the distance from $y_{i,j}$ to any other vertex is more than $2n^2$. This already uses κ^2 of the available $5\kappa^2$ centers. Since there are κ^2 cycles and only $4\kappa^2$ remaining available centers, we proved that each cycle $O_{i,j}$ contains exactly four centers, and no other centers exist in the graph $G_{\mathcal{I}}$. Let $C^O_{i,j}$ be the set of four centers contained in $O_{i,j}$. As each center of $C^O_{i,j}$ covers at most $4n^2 + 1$ vertices of $O_{i,j}$ by balls of radius at most $2n^2$, to cover all $16n^2 + 4$ vertices of $O_{i,j}$ these four centers must be equidistant with distance exactly $4n^2 + 1$ between consecutive centers on $O_{i,j}$. Furthermore, since $\ell_a, \ell'_a, \ell_b, \ell'_b > 2n^2 - 1$ and each edge of $O_{i,j}$ has length 1, to cover $x^h_{i,j}$ for any $h \in [4]$ some center of $C^O_{i,j}$ must lie on a vertex of $O_{i,j}$ adjacent to $x^h_{i,j}$. This means that the four centers of $C^O_{i,j}$ are exactly those vertices $v_{\tau+(h-1)(4n^2+1)}$ corresponding to element s_τ of $S_{i,j}$.

It remains to show that the elements corresponding to the centers in $\bigcup_{i,j\in[\kappa]} C^O_{i,j}$ form a solution to the $\mathrm{GT}_{\leq}$ instance $\mathcal{I}$. For this, consider two neighbouring gadgets $G_{i,j}$ and $G_{i,j+1}$, and let $(a,b) \in S_{i,j}$ and $(a',b') \in S_{i,j+1}$ be the respective elements corresponding to the center sets $C^O_{i,j}$ and $C^O_{i,j+1}$. Note that for any $\hat{b} \in [n]$ we have $\ell_b \leq \ell_{\hat{b}} + 1$ and $\ell'_{b'} \leq \ell'_{\hat{b}} + 1$. Since every edge of the cycles $O_{i,j}$ and $O_{i,j+1}$ has length 1, this means that the distance from the closest centers $v \in C^O_{i,j}$ and $v' \in C^O_{i,j+1}$ to $x^2_{i,j}$ and $x^4_{i,j+1}$, respectively, is determined by the edges of length ℓ_b and $\ell'_{b'}$ incident to v and v', respectively. In particular, the distance between v and v' is $\ell_b + 1 + \ell'_{b'}$, as the path P connecting $x^2_{i,j}$ and $x^4_{i,j+1}$ has length 1. Assume now that $b > b'$, which means that $b \geq b' + 1$ since b and b' are integer. Hence this distance is

$$\ell_b + 1 + \ell'_{b'} = 2n^2 + \frac{b}{n+1} - 1 + 1 + 2n^2 - \frac{b'}{n+1} \geq 4n^2 + \frac{1}{n+1}.$$

As the centers v and v' only cover vertices at distance at most $2n^2$ each, while the edges of the path P have length $\frac{1}{n+2} < \frac{1}{n+1}$, there must be some vertex of P that is not covered by the center set. However this contradicts the fact that the centers form a feasible solution with cost at most $2n^2$, and so $b \leq b'$.

An analogous argument can be made for neighbouring gadgets $G_{i,j}$ and $G_{i+1,j}$, so that $a \leq a'$ for the elements $(a, b) \in S_{i,j}$ and $(a', b') \in S_{i+1,j}$ corresponding to the centers in $C^O_{i,j}$ and $C^O_{i+1,j}$, respectively. Thus a solution to $G_{\mathcal{I}}$ of cost at most $2n^2$ implies a solution to $\mathcal{I}$.

3 Properties of the constructed graph

The reduction of Section 2 proves that the k-Center problem is W[1]-hard for parameter k, since the reduction can be done in polynomial time and k is function of κ. Since this function is quadratic, we can also conclude that, under ETH, there is no $f(k) \cdot n^{o(\sqrt{k})}$ algorithm for k-Center. We will now show that the reduction has various additional properties from which we will be able to conclude Theorem 1. First of all we prove that any constructed graph $G_{\mathcal{I}}$ for an instance $\mathcal{I}$ of $\mathrm{GT}_{\leq}$ is planar and has bounded doubling dimension.

▶ **Lemma 6.** *The graph $G_{\mathcal{I}}$ is planar and has doubling dimension at most $\log_2(36) \approx 5.17$ for $n \geq 3$.*

Proof. It is obvious from the construction of the graph $G_{\mathcal{I}}$ that it is planar. To bound its doubling dimension, consider the shortest-path metric on the vertex set $Y = \{y_{i,j} \in V(G_{\mathcal{I}}) \mid i, j \in [\kappa]\}$ given by the distances between these vertices in $G_{\mathcal{I}}$. As these vertices are arranged in a grid-like fashion, this shortest-path metric on Y approximates the L_1-metric. We consider a set of index pairs, for which the corresponding vertices in Y roughly resemble a ball in the shortest-path metric on Y. That is, consider the set of index pairs $A_{i,j}(a) = \{(i', j') \in [\kappa]^2 \mid |i - i'| + |j - j'| \leq a\}$, and let $V_{i,j}(a) \subseteq V(G_{\mathcal{I}})$ contain all vertices of gadgets $G_{i',j'}$ such that $(i', j') \in A_{i,j}(a)$ in addition to the vertices of paths of length 1 connecting these gadgets. We would like to bound the diameter of the graph induced by $V_{i,j}(a)$ in $G_{\mathcal{I}}$, for which we need the following claim, which we will reuse later.

▶ **Claim 7.** *For any gadget $G_{i,j}$ and $h, h' \in [4]$ with $h \neq h'$, the distance between $x^h_{i,j}$ and $x^{h'}_{i,j}$ lies between $7n^2 - 1$ and $8n^2 + 2$.*

Proof. The distance between $x^h_{i,j}$ and $x^{h'}_{i,j}$ is less than $2(2n^2 + 2n^2 + 1) = 8n^2 + 2$, via the path passing through $y_{i,j}$ and the two vertices of $O_{i,j}$ adjacent to $y_{i,j}$, $x^h_{i,j}$, and $x^{h'}_{i,j}$. Note that the shortest path between $x^h_{i,j}$ and $x^{h'}_{i,j}$ inside the gadget $G_{i,j}$ does not necessarily pass through $y_{i,j}$, but may pass along the cycle $O_{i,j}$ instead. This is because the set $S_{i,j}$ of the $\mathrm{GT}_{\leq}$ instance may contain up to n^2 elements, which would imply a direct edge from $x^h_{i,j}$ to $v_{n^2+(h-1)(4n^2+1)}$ on $O_{i,j}$. Thus we can give a lower bound of $2(2n^2 - 1) + 3n^2 + 1 = 7n^2 - 1$ for the distance between $x^h_{i,j}$ and $x^{h'}_{i,j}$ inside of $G_{i,j}$. This is also the shortest path between these vertices in $G_{\mathcal{I}}$, since any other path needs to pass through at least three gadgets. ◀

By Claim 7, the diameter of $V_{i,j}(a)$ is at most $(8n^2 + 3)(2a + 1)$ as one needs to traverse at most $2a + 1$ gadgets $G_{i',j'}$ with $(i', j') \in A_{i,j}(a)$ and the paths of length 1 connecting them, in order to reach any vertex of $V_{i,j}(a)$ from any other. Assuming $|A_{i,j}(a)| = (2a + 1)^2$, i.e., the set contains the maximum number of index pairs, the diameter of $V_{i,j}(a)$ is at least $7n^2(2a + 1) - 1$, since any path between two points of $V_{i,j}(a)$ at maximum distance must pass through at least $2a + 1$ gadgets $G_{i',j'}$ with $(i', j') \in A_{i,j}(a)$ and the $2a$ paths of length 1 connecting them.

Consider a ball $B_v(2r)$ around a vertex v of radius $2r$ in $G_{\mathcal{I}}$, and let $y_{i,j}$ be the closest vertex of Y to v. The distance between $y_{i,j}$ and v is at most $2(2n^2 + 1) = 4n^2 + 2$, whether v lies on $O_{i,j}$ or on one of the paths of length 1 connecting $G_{i,j}$ with an adjacent gadget. Hence the ball $B_v(2r)$ is contained in a ball of radius $4n^2 + 2 + 2r$ around $y_{i,j}$. The latter ball

is in turn contained in the $V_{i,j}(a)$ set centered at $y_{i,j}$ if its diameter is at most the diameter of $V_{i,j}(a)$, i.e., $2(4n^2+2+2r) \leq 7n^2(2a+1)-1$. This for instance is true if $2a+1 = \frac{r}{n^2}$, $r \geq 4n^2+2$, and $n \geq 1$. From the upper bound on the diameter of a set $V_{i',j'}(a')$, we also know that $V_{i',j'}(a')$ is contained in a ball of radius r around $y_{i',j'}$ if $(8n^2+3)(2a'+1) \leq 2r$, which is true if $2a'+1 = \frac{2r}{8n^2+3}$. However we also want $V_{i',j'}(a')$ to be non-empty, i.e., $a' \geq 0$, which by the latter equality means that $r \geq 4n^2+3/2$. We may cover all vertices of $A_{i,j}(a)$ with $\lceil \frac{2a+1}{2a'+1} \rceil^2$ sets $A_{i',j'}(a')$, since in Y these sets correspond to "squares rotated by 45 degrees of diameter $2a+1$ and $2a'+1$, respectively". Thus we can cover $V_{i,j}(a)$ with $\lceil \frac{2a+1}{2a'+1} \rceil^2$ sets $V_{i',j'}(a')$, i.e., if $r \geq 4n^2+2$ and $n \geq 2$ we can cover a ball of radius $2r$ in $G_{\mathcal{I}}$ with $\lceil \frac{2a+1}{2a'+1} \rceil^2 = \lceil 4 + \frac{3}{2n^2} \rceil^2 \leq 25$ balls of radius r.

$G_{i,j}$ gadgets $y_{i,j}$. is $r < 4+1)^2 = 81$ cover a

If $r < 4n^2+2$, a ball $B_v(2r)$ has radius less than $8n^2+4$. Consider the case when v lies in a gadget $G_{i,j}$ of $G_{\mathcal{I}}$. The distance from $G_{i,j}$ to any cycle $O_{i',j'}$ for which $|i-i'|+|j-j'| \geq 2$ is at least $7n^2-1+2n^2-1 \geq 8n^2+4$, as $n \geq 3$: to reach $O_{i',j'}$ a path from $G_{i,j}$ first needs to traverse a neighbouring gadget of $G_{i,j}$, which we know has diameter at least $7n^2-1$ by Claim 7, and the vertex $x^h_{i',j'}$ has distance more than $2n^2-1$ from $O_{i',j'}$. Thus $B_v(2r)$ contains at most the four neighbouring gadgets of $G_{i,j}$ and the paths of length 1 connected to these. On each of the five cycles $O_{i',j'}$ that intersect $B_v(2r)$, at most 3 balls of radius r are needed to cover all vertices in the intersection of $O_{i',j'}$ and $B_v(2r)$: as the edges of $O_{i',j'}$ all have length 1 we may choose 3 vertices equidistantly at every $\lfloor 2r \rfloor$-th vertex on the part of $O_{i',j'}$ in $B_v(2r)$. As long as $r \geq 1$, any path of length 1 that intersects $B_v(2r)$ can be covered by one ball of radius r. Any vertex $y_{i',j'}$ that lies in $B_v(2r)$ can also be covered by one ball of radius r. As $B_v(2r)$ intersects at most 5 cycles $O_{i',j'}$, as well as at most 5 vertices $y_{i',j'}$, and at most 16 paths of length 1, these amount to at most 36 balls of radius r.

If v does not lie in any gadget $G_{i,j}$, then it lies on some path of length 1 connecting two gadgets. Given that $r < 4n^2+2$, in this case the ball $B_v(2r)$ intersects at most 2 cycles $O_{i,j}$ and vertices $y_{i,j}$, and at most 7 paths of length 1, since by Claim 7 the diameter of a gadget is at least $7n^2-1 \geq r$ if $n \geq 1$. Thus in this case at most 17 balls of radius r suffice.

Finally, if $r < 1$, then a ball $B_v(2r)$ contains only a subpath of some cycle $O_{i,j}$, a subpath of a path of length 1 connecting two gadgets, or a single vertex $y_{i,j}$, since any edge connecting a cycle $O_{i,j}$ to $y_{i,j}$ or some $x^h_{i,j}$ has length more than $2n^2 \geq 1$ if $n \geq 1$. In this case at most 3 balls of radius r suffice to cover all vertices of $B_v(2r)$. ◀

We next show that we can bound the parameters t and h, i.e. the treewidth and highway dimension of $G_{\mathcal{I}}$, as a function of the parameter $k = \Theta(\kappa^2)$. Note that the following lemma bounds the highway dimension in terms of k, no matter how restrictive we make Definition 4 by increasing the constant c.

▶ **Lemma 8.** *For any constant c of Definition 4, the graph $G_{\mathcal{I}}$ has highway dimension at most $O(\kappa^2)$.*

Proof. For any scale $r \in \mathbb{R}^+$ and universal constant $c \geq 4$ we will define a hub set $H_r \subseteq V$ hitting all shortest paths of length more than r in $G_{\mathcal{I}}$, such that $|H_r \cap B_{cr}(v)| = O(\kappa^2)$ for any ball $B_{cr}(v)$ of radius cr in $G_{\mathcal{I}}$. This bounds the highway dimension to $O(\kappa^2)$ according to Definition 4.

Let $X = \{y_{i,j}, x^h_{i,j} \mid h \in \{1,2,3,4\} \wedge i,j \in [\kappa]\}$, i.e. it contains all vertices connecting gadgets $G_{i,j}$ to each other in addition to the vertices $y_{i,j}$. If $r > 8n^2+2$ then $H_r = X$. Any shortest path containing only vertices of a cycle $O_{i,j}$ has length at most $8n^2+2$, since the cycle has length $16n^2+4$. Any (shortest) path that is a subpath of a path connecting two gadgets has length at most 1. Hence any shortest path of length more than $8n^2+2$ must

contain some vertex of X. The total size of X is $5\kappa^2$, and so any ball, no matter its radius, also contains at most this many hubs of H_r.

If $1 \leq r \leq 8n^2 + 2$ then any path of length more than r but not containing any vertex of X must lie on some cycle $O_{i,j} = (v_1, v_2, \ldots, v_{16n^2+4}, v_1)$. We define the set $W_{i,j} = \{v_{1+\lambda\lfloor r \rfloor} \in V(O_{i,j}) \mid \lambda \in \mathbb{N}_0\}$, i.e. it contains every r-th vertex on the cycle after rounding down. This means that every path on $O_{i,j}$ of length more than r contains a vertex of $W_{i,j}$. Thus for these values of r we set $H_r = X \cup \bigcup_{i,j \in [\kappa]} W_{i,j}$. Any ball $B_{cr}(v)$ of radius cr contains $O(c)$ hubs of any $W_{i,j}$. By Claim 7, the distance between any pair of the four vertices $x^h_{i,j}$, where $h \in \{1, 2, 3, 4\}$, that connect a gadget $G_{i,j}$ to other gadgets, is more than $7n^2 - 1$. This means that $B_{cr}(v)$ can only intersect $O(c^2)$ gadgets, since $cr \leq c(8n^2 + 2)$ and the gadgets are connected in a grid-like fashion. Hence the ball $B_{cr}(v)$ only contains $O(c)$ hubs for each of the $O(c^2)$ sets $W_{i,j}$ for which $B_{cr}(v)$ intersect the respective gadget $G_{i,j}$. At the same time each gadget contains only 5 vertices of X. Thus if c is a constant, then the number of hubs of H_r in $B_{cr}(v)$ is constant.

If $r < 1$, a path of length more than r may be a subpath of a path connecting two gadgets. Let $P_{i,j}$ be the path connecting $x^2_{i,j}$ and $x^4_{i,j+1}$ for $j \leq \kappa - 1$, and let $P'_{i,j}$ be the path connecting $x^3_{i,j}$ and $x^1_{i+1,j}$ for $i \leq \kappa - 1$. Recall that each of these paths consists of $n+2$ edges of length $\frac{1}{n+2}$ each. If $P_{i,j} = (u_0, u_1, \ldots, u_{n+2})$, we define the set $U_{i,j} = \{u_{\lambda\lfloor r \rfloor (n+2)} \in V(P_{i,j}) \mid \lambda \in \mathbb{N}_0\}$, and if $P'_{i,j} = (u_0, u_1, \ldots, u_{n+2})$, we define the set $U'_{i,j} = \{u_{\lambda\lfloor r \rfloor (n+2)} \in V(P'_{i,j}) \mid \lambda \in \mathbb{N}_0\}$, i.e. these sets contain vertices of consecutive distance r on the respective paths, after rounding down. Now let $H_r = \bigcup_{i,j \in [\kappa]} V(G_{i,j}) \cup U_{i,j} \cup U'_{i,j}$, so that every path of length more than r contains a hub of H_r. Any ball $B_{cr}(v)$ of radius $cr < c$ intersects only $O(c^2)$ gadgets $G_{i,j}$, as observed above. As the edges of a cycle $O_{i,j}$ have length 1, the ball B contains only $O(c)$ vertices of $O_{i,j}$. Thus $B_{cr}(v)$ contains $O(c)$ hubs of $V(G_{i,j}) \cup U_{i,j} \cup U'_{i,j}$ for each of the $O(c^2)$ gadgets $G_{i,j}$ it intersects. For constant c, this proves the claim. ◀

To bound the treewidth of $G_{\mathcal{I}}$ we use so-called *cops and robber games*. Given a graph G and $\tau \in \mathbb{N}$, a *state* of the τ *cops and robber game* on G is a pair (K, p) where $K \subseteq V$ with $|K| \leq \tau$, and $p \in V \setminus K$. The set K encodes the positions of the τ cops, while p is the position of the robber. The game proceeds in rounds, where each round $z \in \mathbb{N}_0$ is associated with a state (K_z, p_z). Initially, in round 0 the cops first choose positions K_0 and then the robber chooses a position $p_0 \in V \setminus K_0$. In each round $z \geq 1$, first the cops choose a new position K_z, after which the robber can choose a position $p_z \in V \setminus K_z$, such that p_z and p_{z-1} lie in the same connected component of $G - (K_z \cap K_{z-1})$. The cops win the game if after a finite number of rounds, the robber has no position to choose, i.e., the robber is caught. By [26], a graph G has treewidth t if and only if $t + 1$ cops can win in G.

▶ **Lemma 9.** *The graph $G_{\mathcal{I}}$ has treewidth at most $\kappa + O(1)$.*

Proof. We prove that $\kappa + O(1)$ cops can win the cops and robber game on $G_{\mathcal{I}}$. For each $i, j \in [\kappa]$ we define the sets $X^2_{i,j} = \{x^2_{i',j} \mid i' \in [i]\}$ and $X^4_{i,j} = \{x^4_{i',j} \mid i' \in [\kappa] \setminus [i-1]\}$ and let $K_{i,j} = \{y_{i,j}, x^1_{i,j}, x^3_{i,j}\} \cup X^2_{i,j} \cup X^4_{i,j}$ be a position for the cops. The initial position of the cops is $K_{1,1}$, and they will sweep the graph $G_{\mathcal{I}}$ "from left to right" with increasing index j. More precisely we will describe a strategy, which uses a finite number of intermediate rounds to go from position $K_{i,j}$ to position $K_{i+1,j}$ for each $i \leq \kappa - 1$, and from position $K_{\kappa,j}$ to position $K_{1,j+1}$ for each $j \leq \kappa - 1$. After reaching position $K_{\kappa,\kappa}$ the robber will be caught.

Consider a position $K_{i,j}$ and note that the three connected components left after removing all vertices of $K_{i,j}$ from $G_{\mathcal{I}}$ are (a) the cycle $O_{i,j}$, (b) the subgraph $L_{i,j}$ "left of" $G_{\mathcal{I}}$ induced by all gadgets $G_{i',j'}$ and paths $P_{i',j'}, P'_{i',j'}$ for which $j' \leq j - 1$ and $i' \leq \kappa$, but also the gadgets $G_{i',j'}$ and paths $P'_{i',j'}$ for which $j' = j$ and $i' \leq i - 1$, and finally (c) the subgraph

$R_{i,j}$ "right of" $G_{\mathcal{I}}$ induced by all gadgets $G_{i',j'}$ and paths $P_{i',j'}, P'_{i',j'}$ for which either $j' = j$ and $i' \geq i+1$, or $j' \geq j+1$ and $i' \leq \kappa$, but also the paths $P_{i',j}$ where $i' \leq i$ and the path $P'_{i,j}$. The robber's position p has to be in one of these subgraphs, and we will describe a strategy of the cops, which guarantees that p is a vertex of $R_{i,j}$ if the robber has not been caught yet. As $R_{\kappa,\kappa}$ is empty, this means that the cops are able to catch the robber eventually.

In the initial position $K_{1,1}$ the position p cannot be in $L_{1,1}$ as this graph is empty. We now show how in any position $K_{i,j}$ the cycle $O_{i,j}$ can be traversed by the cops using only three additional cops to those in $K_{i,j}$, to catch the robber in case his position p is on $O_{i,j}$. If $O_{i,j} = (v_1, v_2, \ldots, v_{16n^2+4}, v_1)$ we define a sequence of positions $K^h = K_{i,j} \cup \{v_1, v_h, v_{h+1}\}$ where $h \in [16n^2+3]$. Note that $v_1, v_{h+1} \in K^h \cap K^{h+1}$ so that $O_{i,j} - (K^h \cap K^{h+1})$ consists of the two paths $Q_1^h = (v_2, v_3, \ldots, v_h)$ and $Q_2^h = (v_{h+2}, v_{h+3}, v_{16n^2+4})$. The former path Q_1^h is empty for $h = 1$ and thus the robber cannot move to a position on Q_1^h throughout the whole sequence in any step h. In the last step $h = 16n^2+3$ however, $Q_2^{16n^2+3}$ is empty, and so the robber would not have any position to choose in this step if p was on Q_2^h on any previous step h. Thus we may assume that the robber's position lies in $R_{i,j}$ by induction.

If the cops are in position $K_{i,j}$ for some $i \leq \kappa - 1$, then they can move to position $K_{i+1,j}$ as follows. The cops first move to position $K_{i,j} \cup K_{i+1,j}$, which contains $\kappa + 6$ vertices. By induction, the position p of the robber is now in $R_{i,j} - K_{i+1,j}$, i.e., p lies either in $R_{i+1,j}$, $O_{i+1,j}$, or $P'_{i,j}$. As described above, the cops can catch the robber if p is on the cycle $O_{i,j}$. A similar strategy can be used to traverse the path $P'_{i,j}$: if $P'_{i,j} = (u_0, u_1, \ldots, u_{n+2})$ then we define a sequence of positions $K^h = K_{i,j} \cup K_{i+1,j} \cup \{u_{h-1}, u_h\}$ for $h \in [n+1]$, so that $P'_{i,j} - (K^h \cap K^{h+1})$ consists of the two paths $Q_1^h = (u_1, u_2, \ldots, u_{h-1})$ and $Q_2^h = (u_{h+1}, u_{h+2}, \ldots, u_{n+1})$, since $u_0 = x^3_{i,j} \in K_{i,j}$ and $u_{n+2} = x^1_{i+1,j} \in K_{i+1,j}$. As Q_1^1 is empty and $u_h \in K^h \cap K^{h+1}$, the robber's position cannot be on Q_1^h in any subsequent step h, and as Q_2^{n+1} is empty, the robber is caught if p lies on path $P'_{i,j}$. Thus the only possibility left for the robber not to be caught is to be in $R_{i+1,j}$. In this case the cops can switch to position $K_{i+1,j}$ and continue their chase.

Finally the cops also need a strategy to move from position $K_{\kappa,j}$ to position $K_{1,j+1}$ for each $j \leq \kappa - 1$. For this we define the intermediate positions $K'_{i,j} = X^2_{i,j} \cup X^4_{i,j}$, which contain $\kappa + 1$ cops each. Note that $R_{\kappa,j}$ is a connected component of $G_{\mathcal{I}} - (K_{\kappa,j} \cap K'_{\kappa,j})$. Thus the cops can safely switch from position $K_{\kappa,j}$ to $K'_{\kappa,j}$, as the robber is in $R_{\kappa,j}$ by induction. For any $i \in [\kappa]$, removing the vertices of $K'_{i,j}$ from $G_{\mathcal{I}}$ leaves three connected components of which one is $P_{i,j}$ without the endpoints, and one is a component $R'_{i,j}$, which is $R_{i,j}$ without the paths $P_{i',j}$ where $i' \geq i$. By induction the robber's position p is in one of these two components. If p is on $P_{i,j}$, the above strategy for paths shows how to catch the robber. Thus the only possibility left is that p lies in $R'_{i,j}$. If $i \geq 2$, the cops can switch to position $K'_{i-1,j}$, after which p again lies either on $P_{i-1,j}$ or in $R'_{i-1,j}$. For $i = 1$, note that $L_{1,j+1}$ is a connected component of $G_{\mathcal{I}} - (K_{1,j+1} \cap K'_{1,j})$, which by induction does not contain the robber. Thus when switching from position $K'_{1,j}$ to $K_{1,j+1}$ the robber must either be on $O_{1,j+1}$ or $R_{1,j+1}$. We know how to catch the robber if he happens to be on $O_{1,j+1}$ using only three additional cops, and so the only case left is that p is in $R_{1,j+1}$.

Each of the used positions for the cops has at most $\kappa + O(1)$ vertices, and thus the treewidth of $G_{\mathcal{I}}$ is bounded by the same term. ◀

The reduction given in Section 2 together with Lemmas 6, 8 and 9 imply Theorem 1, since the $\mathrm{GT}_{\leq}$ problem is W[1]-hard [8] for parameter κ, and we may assume w.l.o.g. that $n \geq 3$. Moreover $\kappa = \Theta(k)$ and, under ETH, $\mathrm{GT}_{\leq}$ has no $f(\kappa) \cdot n^{o(\kappa)}$ time algorithm [8] for any computable function f.

4 An algorithm for low doubling metrics

In this section we give a simple algorithm that generalizes one from [4], which for D-dimensional L_q metrics compute a $(1+\varepsilon)$-approximation in time $f(\varepsilon,k,D)\cdot n^{O(1)}$. In particular, any such metric has doubling dimension $O(D)$. Here we assume that the input metric has doubling dimension d. A fundamental observation about metrics of bounded doubling dimension is the following, which can be proved by a simple recursive application of Definition 2. Here the aspect ratio is the diameter of X divided by the minimum distance of the metric.

▶ **Lemma 10** ([17]). *Let (X,dist) be a metric with doubling dimension d and $Y\subseteq X$ be a subset with aspect ratio α. Then $|Y|\leq 2^{d\lceil\log_2\alpha\rceil}$.*

To compute a $(1+\varepsilon)$-approximation to k-Center given a graph G with vertex set V, we first compute its shortest-path metric (V,dist). We then compute a *net* of this metric, which is defined as follows.

▶ **Definition 11.** For a metric (X,dist), a subset $Y\subseteq X$ is called a *δ-cover* if for every $u\in X$ there is a $v\in Y$ such that $\mathrm{dist}(u,v)\leq\delta$. A *$\delta$-net* is a δ-cover with the additional property that $\mathrm{dist}(u,v)>\delta$ for all vertices $u,v\in Y$.

Note that a δ-net can be computed greedily in polynomial time. The first step of our algorithm is to guess the optimum cost ρ by trying each of the $\binom{n}{2}$ possible values. For each guess we compute an $\frac{\varepsilon\rho}{2}$-net $Y\subseteq X$. We know that the metric (V,dist) can be covered by k balls of diameter 2ρ each, which means that the aspect ratio of Y inside of each ball is at most $4/\varepsilon$. Thus by Lemma 10, each ball contains $1/\varepsilon^{O(d)}$ vertices of Y, and so $|Y|\leq k/\varepsilon^{O(d)}$.

An optimum k-Center solution $C\subseteq Y$ for (Y,dist) can be computed by brute force in $\binom{|Y|}{k}=k^k/\varepsilon^{O(kd)}$ steps. Since every center of the optimum solution $C^*\subseteq V$ of the input graph has a net point of Y at distance at most $\frac{\varepsilon\rho}{2}$, there exists a k-Center solution in Y of cost at most $(1+\varepsilon/2)\rho$. The computed center set $C\subseteq Y$ thus also has cost at most $\frac{\varepsilon\rho}{2}$. Therefore C covers all of V with balls of radius $(1+\varepsilon)\rho$, since every vertex of V is at distance $\frac{\varepsilon\rho}{2}$ from some vertex of Y. Thus C is a $(1+\varepsilon)$-approximation of the input graph. Considering the guessed values of ρ in increasing order, outputting the first computed solution with at most k centers gives the algorithm of Theorem 3.

References

1 I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM*, 63(5):41, 2016.

2 I. Abraham, D. Delling, A. Fiat, A.V. Goldberg, and R.F. Werneck. VC-dimension and shortest path algorithms. In *ICALP*, pages 690–699, 2011.

3 I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*, pages 782–793, 2010.

4 P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

5 H. Bast, S. Funke, and D. Matijevic. Ultrafast shortest-path queries via transit nodes. *9th DIMACS Implementation Challenge*, 74:175–192, 2009.

6 H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortest-path queries in road networks. In *ALENEX*, pages 46–59, 2007.

7 A. Becker, P. N. Klein, and D. Saulpic. Polynomial-time approximation schemes for k-center and bounded-capacity vehicle routing in metrics with bounded highway dimension. *ArXiv e-prints, arXiv:1707.08270 [cs.DS]*, 2017.

8 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *Transactions on Algorithms*, 1(1):33–47, 2005.

10 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*. ACM Press, 2014.

11 R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.

12 David Eisenstat, Philip N Klein, and Claire Mathieu. Approximating k-center in planar graphs. In *SODA*, pages 617–627, 2014.

13 T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *STOC*, pages 434–444, 1988.

14 A. E. Feldmann, W. S. Fung, J. Könemann, and I. Post. A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. In *ICALP*, pages 469–480, 2015.

15 A. E. Feldmann, W. S. Fung, J. Könemann, and I. Post. A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. *ArXiv preprint arXiv:1502.04588*, 2015.

16 Andreas Emil Feldmann. Fixed parameter approximations for k-center problems in low highway dimension graphs. In *ICALP*, pages 588–600. Springer Berlin Heidelberg, 2015.

17 A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543, 2003.

18 D. S. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

19 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. In *ISAAC*, pages 50:1–50:13, 2017.

20 P. Klein. Personal communication, 2017.

21 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *STOC*, pages 224–237. ACM Press, 2017.

22 D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

23 Dániel Marx. Efficient approximation schemes for geometric problems? In *European Symposium on Algorithms*, pages 448–459. Springer, 2005.

24 Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *ESA*, pages 865–877. Springer, 2015.

25 J. Plesník. On the computational complexity of centers locating in a graph. *Aplikace matematiky*, 25(6):445–452, 1980.

26 Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

27 Karthik Srikanta, Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *arXiv preprint*, abs/1711.11029, 2017.

28 V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

29 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

Algorithms for the Discrete Fréchet Distance Under Translation

Omrit Filtser[1]
Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
omritna@cs.bgu.ac.il

Matthew J. Katz[2]
Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Abstract

The (discrete) Fréchet distance (DFD) is a popular similarity measure for curves. Often the input curves are not aligned, so one of them must undergo some transformation for the distance computation to be meaningful. Ben Avraham et al. [5] presented an $O(m^3n^2(1+\log(n/m))\log(m+n))$-time algorithm for DFD between two sequences of points of sizes m and n in the plane under translation. In this paper we consider two variants of DFD, both under translation.

For DFD with shortcuts in the plane, we present an $O(m^2n^2\log^2(m+n))$-time algorithm, by presenting a dynamic data structure for reachability queries in the underlying directed graph. In 1D, we show how to avoid the use of parametric search and remove a logarithmic factor from the running time of (the 1D versions of) these algorithms and of an algorithm for the weak discrete Fréchet distance; the resulting running times are thus $O(m^2n(1+\log(n/m)))$, for the discrete Fréchet distance, and $O(mn\log(m+n))$, for its two variants.

Our 1D algorithms follow a general scheme introduced by Martello et al. [21] for the Balanced Optimization Problem (BOP), which is especially useful when an efficient dynamic version of the feasibility decider is available. We present an alternative scheme for BOP, whose advantage is that it yields efficient algorithms quite easily, without having to devise a specially tailored dynamic version of the feasibility decider. We demonstrate our scheme on the *most uniform path* problem (significantly improving the known bound), and observe that the weak DFD under translation in 1D is a special case of it.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases curve similarity, discrete Fréchet distance, translation, algorithms, BOP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.20

1 Introduction

Polygonal curves play an important role in many applied domains, and it is a challenging task to compare them in a way that will reflect our intuitive notion of resemblance. The *Fréchet distance* is a useful and well studied similarity measure that has been applied in many diverse settings. Consider a man and a dog connected by a leash, each walking along a curve. They can control their speed but they are not allowed to backtrack. The Fréchet

[1] O. Filtser was supported by the Ministry of Science, Technology & Space, Israel, and by the Lynn and William Frankel Center.
[2] M. Katz was supported by grant 1884/16 from the Israel Science Foundation and grant 2014/170 from the US-Israel Binational Science Foundation.

distance between the two curves is the minimum length of a leash that is sufficient for such a dog-walk from the starting points to the end points of the curves.

Intuitively, the *discrete Fréchet distance* (DFD) replaces the curves by two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, and replaces the man and dog by two frogs (connected by a leash), the A-frog and the B-frog, initially placed at a_1 and b_1, respectively. At each move, the A-frog or the B-frog (or both) jumps from its current point to the next one. We are interested in the minimum length of a leash that allows the A-frog and the B-frog to reach a_n and b_m, respectively. The discrete distance is considered a good approximation of the continuous one, and is somewhat easier to compute. Both versions of the Fréchet distance (continuous and discrete) can be computed in roughly $O(n^2)$-time [1, 2, 7, 8, 14].

When the curves or the sampled sequences of points are generated by physical sensors, such as GPS devices, inaccurate measurements may occur. Since the Fréchet distance is a bottleneck measure and is thus very sensitive to outliers, several variants for handling outliers have been proposed, among these are: average and summed Fréchet distance [6, 10, 13], partial Fréchet similarity [9], and Fréchet distance with shortcuts [4, 11, 12].

In the *(one-sided) discrete Fréchet distance with shortcuts* (DFDS), we allow the A-frog to jump to any point that comes later in its sequence, rather than to only the next point. The B-frog has to visit all the B points in order, as in the standard discrete Fréchet distance. Driemel and Har-Peled [12] introduced the (continuous) Fréchet distance with shortcuts. They considered the vertex-restricted version where the dog is allowed to take shortcuts only by walking from a vertex v to any succeeding vertex w along the line segment connecting v and w, and presented an $O(n^5 \log n)$-time algorithm for this version. Later, Buchin et al. [11] showed that in the general case, where the dog is allowed to take shortcuts between any two points on its (continuous) curve, the problem becomes NP-hard. In the discrete case, however, the situation is much better. Ben Avraham et al. [4] presented an $O((m+n)^{6/5+\epsilon})$ expected-time randomized algorithm for the problem. Moreover, they showed that the decision version in this case can be solved in linear time.

Another well-known variant of the Fréchet distance is the *weak discrete Fréchet distance* (WDFD), in which the frogs are allowed to jump also backwards to the previous point in their sequence. Alt and Godau [2] showed that the continuous weak Fréchet distance can be computed in $O(mn \log(mn))$ time.

In many applications, the input curves are not necessarily aligned, and one of them needs to be adjusted (i.e., undergo some transformation) for the distance computation to be meaningful. In the *discrete Fréchet distance under translation*, we are given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, and wish to find a translation t that minimizes the discrete Fréchet distance between A and $B + t$.

For points in the plane, Alt et al. [3] gave an $O(m^3n^3(m+n)^2 \log(m+n))$-time algorithm for computing the continuous Fréchet distance under translation, and an algorithm computing a $(1+\epsilon)$-approximation in $O(\epsilon^{-2}mn)$ time. In 3D, Wenk [24] showed that the minimum continuous Fréchet distance under any reasonable family of transformations, can be computed in $O((m+n)^{3f+2} \log(m+n))$ time, where f is the number of degrees of freedom for moving one sequence w.r.t. the other. For translations only ($f = 3$), the minimum continuous Fréchet distance in $\mathbb{R}^3$ can be computed in $O((m+n)^{11} \log(m+n))$ time.

In the discrete case, the situation is a little better. For points in the plane, Jiang et al. [20] gave an $O(m^3n^3 \log(m+n))$-time algorithm for DFD under translation, and an $O(m^4n^4 \log(m+n))$-time algorithm when both rotations and translations are allowed. Mosig et al. [22] presented an approximation algorithm for DFD under translation, rotation and scaling in the plane, with approximation factor close to 2 and running time $O(m^2n^2)$. Finally, Ben Avraham et al. [5] presented an $O(m^3n^2(1+\log(n/m)) \log(m+n))$-time algorithm for DFD under translation.

1.1 Preliminaries

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points. We define a directed graph $G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$, whose vertices are the possible positions of the frogs and whose edges are the possible moves between positions:
$E_A = \{\langle (a_i, b_j), (a_{i+1}, b_j)\rangle\}, E_B = \{\langle (a_i, b_j), (a_i, b_{j+1})\rangle\}, E_{AB} = \{\langle (a_i, b_j), (a_{i+1}, b_{j+1})\rangle\}$.

The set E_A corresponds to moves where only the A-frog jumps forward, the set E_B corresponds to moves where only the B-frog jumps forward, and the set E_{AB} corresponds to moves where both frogs jump forward. Notice that any valid sequence of moves (with unlimited leash length) corresponds to a path in G from (a_1, b_1) to (a_n, b_m), and vice versa.

It is likely that not all positions in $A \times B$ are **valid**; for example, when the leash is short. We thus assume that we are given an **indicator** function $\sigma : A \times B \to \{0, 1\}$, which determines for each position whether it is valid or not. Now, we say that a position (a_i, b_j) is a **reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), consisting of only valid positions, i.e., for each position $(a_k, b_l) \in P$, we have $\sigma(a_k, b_l) = 1$.

Let $d(a_i, b_j)$ denote the Euclidean distance between a_i and b_j. For any distance $\delta \geq 0$, the function σ_δ is defined as follows: $\sigma_\delta(a_i, b_j) = \begin{cases} 1, & d(a_i, b_j) \leq \delta \\ 0, & \text{otherwise} \end{cases}$.

The **discrete Fréchet distance** $d_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a reachable position w.r.t. σ_δ.

One-sided shortcuts. Let σ be an indicator function. We say that a position (a_i, b_j) is an **s-reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that $\sigma(a_1, b_1) = 1$, $\sigma(a_i, b_j) = 1$, and for each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$). We call such a path an **s-path**. In general, an s-path consists of both valid and non-valid positions. Consider the sequence S of positions that is obtained from P by deleting the non-valid positions. Then S corresponds to a sequence of moves, where the A-frog is allowed to skip points, and the leash satisfies σ. Since in any path in G the two indices (of the A-points and of the B-points) are monotonically non-decreasing, it follows that in S the B-frog visits each of the points $b_1, \ldots, b_j$, in order, while the A-frog visits only a subset of the points $a_1, \ldots, a_i$ (including a_1 and a_i), in order.

The **discrete Fréchet distance with shortcuts** $d^s_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is an s-reachable position w.r.t. σ_δ.

Weak Fréchet distance. Let $G_w = G(V = A \times B, E_w)$, where $E_w = \{(u, v) | \langle u, v\rangle \in E_A \cup E_B \cup E_{AB}\}$. That is, G_w is an undirected graph obtained from the graph G of the 'strong' version, which contains directed edges, by removing the directions from the edges. Let σ be an indicator function. We say that a position (a_i, b_j) is a **w-reachable position** (w.r.t. σ), if there exists a path P in G_w from (a_1, b_1) to (a_i, b_j) consisting of only valid positions. Such a path corresponds to a sequence of moves of the frogs, with a leash satisfying σ, where backtracking is allowed.

The **weak discrete Fréchet distance** $d^w_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a w-reachable position w.r.t. σ_δ.

The translation problem. Given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, we wish to find a translation t^* that minimizes $d_{dF}(A, B+t)$ (similarly, $d^s_{dF}(A, B+t)$ and $d^w_{dF}(A, B+t)$), over all translations t. Denote $\widehat{d_{dF}}(A, B) = \min_t\{d_{dF}(A, B+t)\}$, $\widehat{d^s_{dF}}(A, B) = \min_t\{d^s_{dF}(A, B+t)\}$ and $\widehat{d^w_{dF}}(A, B) = \min_t\{d^w_{dF}(A, B+t)\}$.

1.2 Our results

As mentioned earlier, Ben Avraham et al. [5] presented an algorithm that computes DFD under translation in $O(m^3n^2(1+\log(n/m))\log(m+n))$ time. Given sequences A and B and an indicator function σ, they construct a dynamic data structure in $O(mn)$ time (which also stores the information whether (a_n, b_m) is reachable or not). Following a single change (i.e., some valid position becomes non-valid or vice versa), the data structure can be updated in $O(m(1+\log(n/m)))$ time.

Our first major result is an efficient algorithm for DFDS under translation. We provide a dynamic data structure which supports updates in $O(\log(m+n))$ time per update, where in an update the value of σ for some position (a_i, b_j) changes from valid to non-valid or vice versa. Following an update, one can determine whether the final position (a_n, b_m) is reachable from the starting position (a_1, b_1), with shortcuts, in $O(\log(m+n))$ time. The data structure is based on Sleator and Tarjan's Link-Cut Trees structure [23], and, by plugging it into the optimization algorithm of Ben Avraham et al. [5], we obtain an $O(m^2n^2\log^2(m+n))$-time algorithm for DFDS under translation; an order of magnitude faster than the the algorithm for DFD under translation.

In 1D, the optimization algorithm of [5] yields an $O(m^2n(1+\log(n/m))\log(m+n))$-time algorithm for DFD, using their reachability structure, an $O(mn\log^2(m+n))$-time algorithm for DFDS, using our reachability with shortcuts structure, and an $O(mn\log^2(m+n))$-time algorithm for WDFD, using a reachability structure of Eppstein et al. [15] for undirected planar graphs. We describe a simpler optimization algorithm for 1D, which avoids the need for parametric search and yields an $O(m^2n(1+\log(n/m)))$-time algorithm for DFD and $O(mn\log(m+n))$-time algorithms for DFDS and WDFD; i.e., we remove a logarithmic factor from the bounds obtained with the algorithm of Ben Avraham et al.

Our optimization algorithm for 1D follows a general scheme introduced by Martello et al. [21] for the Balanced Optimization Problem (BOP). BOP is defined as follows. Let $E = \{e_1, \ldots, e_l\}$ be a set of l elements (where here $l = O(mn)$), $c : E \to \mathbb{R}$ a cost function, and $\mathcal{F}$ a set of feasible subsets of E. Find a feasible subset $S^* \in \mathcal{F}$ that minimizes $\max\{c(e_i) : e_i \in S\} - \min\{c(e_i) : e_i \in S\}$, over all $S \in \mathcal{F}$. Given a feasibility decider that decides whether a subset is feasible or not in $f(l)$ time, the algorithm of [21] finds an optimal range in $O(lf(l) + l\log l)$-time.

The scheme of [21] is especially useful when an efficient dynamic version of the feasibility decider is available, as in the case of DFD (where $f(l) = O(m(1+\log(n/m)))$), DFDS (where $f(l) = O(\log(m+n))$), and WDFD (where $f(l) = O(\log(m+n))$).

Our second major result is an alternative scheme for BOP. Our optimization scheme does not require a specially tailored dynamic version of the feasibility decider in order to obtain faster algorithms (than the naive $O(lf(l) + l\log l)$ one), rather, whenever the underlying problem has some desirable properties, it produces algorithms with running time $O(f(l)\log^2 l + l\log l)$. Thus, the advantage of our scheme is that it yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider, a task which is often difficult if at all possible.

We demonstrate our scheme on the *most uniform path* problem (MUPP). Given a weighted graph $G = (V, E, w)$ with n vertices and m edges and two vertices $s, t \in V$, the goal is to find a path P^* in G between s and t that minimizes $\max\{w(e) : e \in P\} - \min\{w(e) : e \in P\}$, over all paths P from s to t. This problem was introduced by Hansen et al. [18], who gave an $O(m^2)$-time algorithm for it. By using a dynamic connectivity data structure of Holm et al. [19], one can reduce the running time to $O(m\log^2 n)$. We apply our scheme to MUPP to obtain a much simpler algorithm with the same ($O(m\log^2 n)$) running time. Finally, we

observe that WDFD under translation in 1D can be viewed as a special case of MUPP, so we immediately obtain a much simpler algorithm than the one based on Eppstein et al.'s dynamic data structure (see above), at the cost of an additional logarithmic factor.

2 DFDS under translation

The discrete Fréchet distance (and its shortcuts variant) between A and B is determined by two points, one from A and one from B. Consider the decision version of the translation problem: given a distance δ, decide whether $\widehat{d_{dF}}(A,B) \leq \delta$ (or $\widehat{d^s_{dF}}(A,B) \leq \delta$).

Ben Avaraham et al. [5] described a subdivision of the plane of translations: given two points $a \in A$ and $b \in B$, consider the disk $D_\delta(a-b)$ of radius δ centered at $a-b$, and notice that $t \in D_\delta(a-b)$ if and only if $d(a-b,t) \leq \delta$ (or $d(a,b+t) \leq \delta$). That is, $D_\delta(a-b)$ is precisely the set of translations t for which $b+t$ is at distance at most δ from a. They construct the arrangement A_δ of the disks in $\{D_\delta(a-b) \mid (a,b) \in A \times B\}$, which consists of $O(m^2n^2)$ cells. Then, they initialize their dynamic data structure for (discrete Fréchet) reachability queries, and traverse the cells of A_δ such that, when moving from one cell to its neighbor, the dynamic data structure is updated and queried a constant number of times in $O(m(1+\log(n/m))$ time. Finally, they use parametric search in order to find an optimal translation, which adds only a $O(\log(m+n))$ factor to the running time.

In this section we present a dynamic data structure for s-reachability queries, which allows updates and queries in $O(\log(m+n))$ time. We observe that the same parametric search can be used in the shortcuts variant, since the critical values are the same. Thus, by combining our dynamic data structure with the parametric search of [5], we obtain an $O(m^2n^2\log^2(m+n))$-time algorithm for DFDS under translation.

We now describe the dynamic data structure for DFDS. Consider the decision version of the problem: given a distance δ, we would like to determine whether $d^s_{dF}(A,B) \leq \delta$, i.e., whether (a_n,b_m) is an s-reachable position w.r.t. σ_δ. In [4], Ben Avraham et al. presented a linear time algorithm for this decision problem. Informally, the decision algorithm on the graph G is as follows: starting at (a_1,b_1), the B-frog jumps forward (one point at a time) as long as possible, while the A-frog stays in place, then the A-frog makes the smallest forward jump needed to allow the B-frog to continue. They continue advancing in this way, until they either reach (a_n,b_m) or get stuck.

Consider the (directed) graph $G_\delta = G(V = A \times B, E = E'_A \cup E'_B)$, where
$E'_A = \{\langle (a_i,b_j),(a_{i+1},b_j)\rangle \mid \sigma_\delta(a_i,b_j) = 0,\ 1 \leq i \leq n-1,\ 1 \leq j \leq m\}$, and
$E'_B = \{\langle (a_i,b_j),(a_i,b_{j+1})\rangle \mid \sigma_\delta(a_i,b_j) = 1,\ 1 \leq i \leq n,\ 1 \leq j \leq m-1\}$.

In G_δ, if the current position of the frogs is valid, only the B-frog may jump forward and the A-frog stays in place. And, if the current position is non-valid, the B-frog stays in place and only the A-frog may jump forward. Let M_δ be an $n \times m$ matrix such that $M_{i,j} = \sigma_\delta(a_i,b_j)$. Each vertex in G_δ corresponds to a cell of the matrix. The directed edges of G_δ correspond to right-moves (the B-frog jumps forward) and upward-moves (the A-frog jumps forward) in the matrix. Any right-move is an edge originating at a valid vertex, and any upward-move is an edge originating at a non-valid vertex (see Figure 1).

▶ **Observation 1.** *G_δ is a set of rooted binary trees, where a root is a vertex of out-degree 0.*

Proof. Clearly, G is a directed acyclic graph, and G_δ is a subgraph of G. In G_δ, each vertex has at most one outgoing edge. It is easy to see (by induction on the number of vertices) that such a graph is a set of rooted trees. ◀

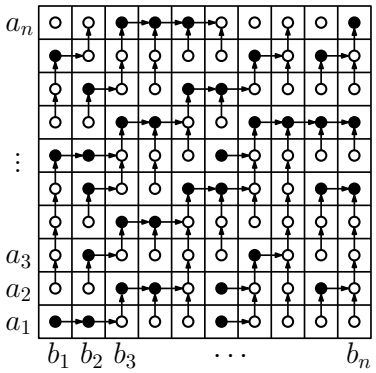


Figure 1 The graph G_δ on the matrix M_δ. The black vertices are valid and the white ones are non-valid.

We call a path P in G from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \le i'$, $j \le j'$, a **partial s-path**, if for each b_l, $j \le l < j'$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma_\delta(a_k, b_l) = 1$).

▶ **Observation 2.** *All the paths in G_δ are partial s-paths.*

Proof. Let P be a path from (a_i, b_j) to $(a_{i'}, b_{j'})$ in G_δ. Each right-move in P advances the B-frog by one step forward. If $j = j'$ then the claim is vacuously true. Else, P must contain a right-move for each b_l, $j \le l < j'$. Any right-move is an edge originating at a valid vertex, thus for any $j \le l < j'$ there exists a position $(a_k, b_l) \in P$ such that $\sigma_\delta(a_k, b_l) = 1$. ◀

Denote by $r(a_i, b_j)$ the root of (a_i, b_j) in G_δ.

▶ **Lemma 3.** *(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \le i \le n$.*

Proof. Assume that $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \le i \le n$. Then by Observation 2 there is a partial s-path from (a_1, b_1) to (a_i, b_m) in G_δ, and since $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$ we have an s-path from (a_1, b_1) to (a_n, b_m).

Now assume that (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. Then, in particular, $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$, and there exists an s-path P in G from (a_1, b_1) to (a_n, b_m). Let P' be the path in G_δ from (a_1, b_1) to $r(a_1, b_1)$. Informally, we claim that P' is always not above P. More precisely, we prove that if a position (a_i, b_j) is an s-reachable position in G, then there exists a position $(a_{i'}, b_j) \in P'$, $i' \le i$, such that $\sigma_\delta(a_{i'}, b_j) = 1$. In particular, since (a_n, b_m) is an s-reachable position in G, there exists a position $(a_{i'}, b_m) \in P'$, $i' \le n$, such that $\sigma_\delta(a_{i'}, b_m) = 1$, and thus $r(a_1, b_1) = (a_{i''}, b_m)$ for some $i' \le i'' \le n$.

We prove this claim by induction on j. The base case where $j = 1$ is trivial, since $(a_1, b_1) \in P \cap P'$ and $\sigma_\delta(a_1, b_1) = 1$. Let P be an s-path from (a_1, b_1) to (a_i, b_{j+1}), then $\sigma_\delta(a_i, b_{j+1}) = 1$. Let (a_k, b_j), $k \le i$, be a position in P such that $\sigma_\delta(a_k, b_j) = 1$. (a_k, b_j) is an s-reachable position in G, so by the induction hypothesis there exists a vertex $(a_{k'}, b_j) \in P'$, $k' \le k$, such that $\sigma_\delta(a_{k'}, b_j) = 1$. By the construction of G_δ, there is an edge $\langle (a_{k'}, b_j), (a_{k'}, b_{j+1}) \rangle$, and we have $(a_{k'}, b_{j+1}) \in P'$. Now, let $k' \le i' \le i$ be the smallest index such that $\sigma_\delta(a_{i'}, b_{j+1}) = 1$. Since there are no right-moves in P' before reaching $(a_{i'}, b_{j+1})$, we have $(a_{i'}, b_{j+1}) \in P'$. ◀

We represent G_δ using the Link-Cut tree data structure, which was developed by Sleator and Tarjan [23]. The data structure stores a set of rooted trees and supports the following operations in $O(\log n)$ amortized time:

- $Link(v, u)$ – connect a root node v to another node u as its child.
- $Cut(v)$ – disconnect the subtree rooted at v from the tree to which it belong.
- $FindRoot(v)$ – find the root of the tree to which v belongs.

Now, in order to maintain the representation of G_δ following a single change in σ_δ (i.e., when switching one position (a_i, b_j) from valid to non-valid or vice versa), one edge should be removed and one edge should be added to the structure. We update our structure as follows: Let T be the tree containing (a_i, b_j).

- When switching (a_i, b_j) from valid to non-valid, we first need to remove the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$, if $j < m$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $i < n$, we add the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_{i+1}, b_j) as its child ($Link((a_i, b_j), (a_{i+1}, b_j))$).
- When switching a position from non-valid to valid, we need to remove the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$, if $i < n$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $j < m$, we add the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_i, b_{j+1}) as its child ($Link((a_i, b_j), (a_i, b_{j+1}))$).

Assume $\sigma_\delta(a_1, b_1) = \sigma_\delta(a_n, b_m) = 1$. By Lemma 3, in the Link-Cut tree data structure representing G_δ, $FindRoot(a_1, b_1)$ is (a_i, b_m) for some $1 \leq i \leq n$ if and only if (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. We thus obtain the following theorem.

▶ **Theorem 4.** *Given sequences A and B and an indicator function σ_δ, one can construct a dynamic data structure in $O(mn \log(m+n))$ time, which supports the following operations in $O(\log(m+n))$ time: (i) change a single value of σ_δ, and (ii) check whether (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ.*

▶ **Theorem 5.** *Given sequences A and B with n and m points respectively in the plane, $\widehat{d^s_{dF}}(A, B)$ can be computed in $O(m^2 n^2 \log^2(m+n))$-time.*

3 Translation in 1D

The algorithm of [5] can be generalized to any constant dimension $d \geq 1$; only the size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$. The running time of the algorithm for two sequences of points in $\mathbb{R}^d$ is therefore $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$, for DFD, and $O(m^d n^d \log^2(m+n))$, for DFDS and WDFD; see relevant paragraph in Section 1.2.

When considering the translation problem in 1D, we can improve the bounds above by a logarithmic factor, by avoiding the use of parametric search and applying a direct approach instead. We thus obtain an $O(m^2 n(1 + \log(n/m)))$-time algorithm, for DFD, and an $O(mn \log(m+n))$-time algorithm, for DFDS and WDFD.

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points in $\mathbb{R}^d$. Consider the set $\mathcal{D} = \{a_i - b_j \mid a_i \in A, b_j \in B\}$. Then, each vertex $v = (a_i, b_j)$ of the graph G has a corresponding point $\overline{v} = (a_i - b_j)$ in $\mathcal{D}$. Given a path P in G from (a_1, b_1) to (a_n, b_m), denote by $\overline{V(P)}$ the set of points of $\mathcal{D}$ corresponding to the vertices $V(P)$ of P. Denote by $S(o, r)$ the sphere with center o and radius r. We define a new indicator function:

$$\sigma_{S(o,r)}(a_i, b_j) = \begin{cases} 1, & d(a_i - b_j, o) \leq r \\ 0, & \text{otherwise} \end{cases}.$$

▶ **Lemma 6.** *Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, t^* is a translation that minimizes $d_{dF}(A, B + t)$, over all translations t, and $d_{dF}(A, B + t^*) = \delta$.*

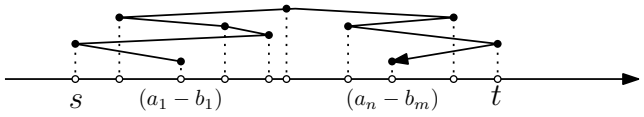


Figure 2 The points of $\overline{V(P)}$.

Proof. Let t be a translation such that $d_{dF}(A, B + t) = \delta'$, and denote $S' = S(t, \delta')$. Thus, there exist a path P from (a_1, b_1) to (a_n, b_m) in G such that for each vertex (a, b) of P, $d(a, b + t) \leq \delta'$. But $d(a, b + t) = d(a - b, t)$, so for each vertex (a, b) of P, $d(a - b, t) \leq \delta'$, and thus (a_n, b_m) is a reachable position w.r.t. $\sigma_{S'}$. Since S is the smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S, we get that $\delta' \geq \delta$.

Now, since (a_n, b_m) is a reachable position w.r.t. σ_S, there exists a path P from (a_1, b_1) to (a_n, b_m), such that for each vertex (a, b) of P, $d(a - b, t^*) \leq \delta$. But again $d(a - b, t^*) = d(a, b + t^*)$, and thus $d_{dF}(A, B + t^*) \leq \delta$. ◀

Notice that the above lemma is true for the shortcuts and the weak variants as well, by letting (a_n, b_m) be an s-reachable or a w-reachable position, respectively.

Thus, our goal is to find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S. We can perform an exhaustive search: check for each sphere S defined by $d + 1$ points of $\mathcal{D}$ whether (a_n, b_m) is a reachable position w.r.t. σ_S. There are $O(m^{d+1}n^{d+1})$ such spheres, and checking whether (a_n, b_m) is a reachable position in G takes $O(mn)$ time. This yields an $O(m^{d+2}n^{d+2})$-time algorithm.

When considering the problem on the line, the goal is to find a path P from (a_1, b_1) to (a_n, b_m), such that the one-dimensional distance between the leftmost point in $\overline{V(P)}$ and the rightmost point in $\overline{V(P)}$ is minimum (see Figure 2). In other words, our indicator function is now defined for a given range $[s, t]$: $\sigma_{[s,t]}(a_i, b_j) = \begin{cases} 1, & s \leq a_i - b_j \leq t \\ 0, & \text{otherwise} \end{cases}$.

We say that a range $[s, t]$ is a **feasible range** if (a_n, b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$. Now, we need to find the smallest feasible range delimited by two points of $\mathcal{D}$.

Consider the following search procedure: Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$. Set $p \leftarrow 1, q \leftarrow 1$. While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p,d_q]}$, set $p \leftarrow p + 1$, else set $q \leftarrow q + 1$. Return the translation corresponding to the smallest feasible range $[d_p, d_q]$ that was found during the while loop.

We use the data structure of [5] for the decision queries, and update it in $O(m(1+\log(n/m))$ time in each step of the algorithm. For DFDS we use our data structure, and for WDFD we use the data structure of [15], where in both the cost of a decision query or an update is $O(\log(m + n))$.

▶ **Theorem 7.** *Let A and B be two sequences of n and m points ($m \leq n$), respectively, on the line. Then, $\widehat{d_{dF}}(A, B)$ can be computed in $O(m^2 n(1 + \log(n/m)))$ time, and $\widehat{d^s_{dF}}(A, B)$ and $\widehat{d^w_{dF}}(A, B)$ can be computed in $O(mn \log(m + n))$ time.*

4 A general scheme for BOP

In the previous section we showed that DFD, DFDS, and WDFD, all under translation and in 1D, can be viewed as BOP. In this section, we present a general scheme for BOP, which yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider.

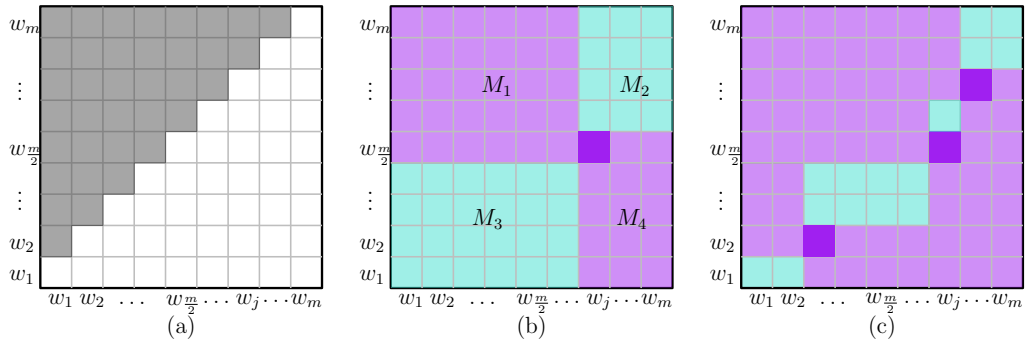


Figure 3 The matrix of possible ranges. (a) The shaded cells are invalid ranges. (b) The cell $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4. (c) The four submatrices at the end of the second level of the recursion tree.

BOP's definition (see Section 1.2) is especially suited for graphs, where, naturally, E is the set of weighted edges of the graph, and $\mathcal{F}$ is a family of well-defined structures, such as matchings, paths, spanning trees, cut-sets, edge covers, etc.

Let $G = (V, E, w)$ be a weighted graph, where V is a set of n vertices, E is a set of m edges, and $w : E \to \mathbb{R}$ is a weight function. Let $\mathcal{F}$ be a set of feasible subsets of E. For a subset $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$. The Balanced Optimization Problem on Graphs (BOPG) is to find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$. A range $[l, u]$ is a **feasible range** if there exists a feasible subset $S \in \mathcal{F}$ such that $w(e) \in [l, u]$ for each $e \in S$. A **feasibility decider** is an algorithm that decides whether a given range is feasible.

We assume for simplicity that each edge has a unique weight. Our goal is to find the smallest feasible range. First, we sort the m edges by their weights, and let $e_1, e_2, \ldots, e_m$ be the resulting sequence. Let $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.

Let M be the matrix whose rows correspond to $w_1, w_2, \ldots, w_m$ and whose columns correspond to $w_1, w_2, \ldots, w_m$ (see Figure 3(a)). A cell $M_{i,j}$ of the matrix corresponds to the range $[w_i, w_j]$. Notice that some of the cells of M correspond to invalid ranges: when $i > j$, we have $w_i > w_j$ and thus $[w_i, w_j]$ is not a valid range.

M is sorted in the sense that range $M_{i,j}$ contains all the ranges $M_{i',j'}$ with $i \leq i' \leq j' \leq j$. Thus, we can perform a binary search in the middle row to find the smallest feasible range $M_{\frac{m}{2},j} = [w_{\frac{m}{2}}, w_j]$ among the ranges in this row. $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4 (see Figure 3(b)). Each of the ranges in M_1 is contained in a range of the middle row which is not a feasible range, hence none of the ranges in M_1 is a feasible range. Each of the ranges in M_4 contains $M_{\frac{m}{2},j}$ and hence is at least as large as $M_{\frac{m}{2},j}$. Thus, we may ignore M_1 and M_4 and focus only on the ranges in the submatrices M_2 and M_3.

Sketch of the algorithm. We perform a recursive search in the matrix M. The input to a recursive call is a submatrix M' of M and a corresponding graph G'. Let $[w_i, w_j]$ be a range in M'. The feasibility decider can decide whether $[w_i, w_j]$ is a feasible range or not by consulting the graph G'. In each recursive call, we perform a binary search in the middle row of M' to find the smallest feasible range in this row, using the corresponding graph G'. Then, we construct two new graphs for the two submatrices of M' in which we still need to search in the next level of the recursion.

Algorithm 1 Balance($G([l, l'] \times [u', u])$)

1. Set $i = \frac{l+l'}{2}$
2. Perform a binary search on the ranges $[i, j]$, $u' \leq j \leq u$, to find the smallest feasible range, using the feasibility decider with the graph $G([l, l'] \times [u', u])$ as input.
3. If there is no feasible range, then:
   a. If $l = l'$, return ∞.
   b. Else, construct $G_1 = G([l, i-1] \times [u', u])$ and return Balance(G_1).
4. Else, let $[w_i, w_j]$ be the smallest feasible range found in the binary search.
   a. If $l = l'$, return $(w_j - w_i)$.
   b. Else, construct two new graphs, $G_1 = G([i+1, l'] \times [j, u])$ and $G_2 = G([l, i-1] \times [u', j-1])$, and return $\min\{(w_j - w_i), \text{Balance}(G_1), \text{Balance}(G_2)\}$.

The number of potential feasible ranges is equal to the number of cells in M, which is $O(m^2)$. But, since we are looking for the smallest feasible range, we do not need to generate all of them. We only use M to illustrate the search algorithm, its cells correspond to the potential feasible ranges, but do not contain any values. We thus represent M and its submatrices by the indices of the sorted list of weights that correspond to the rows and columns of M. For example, we represent M by $M([1, m] \times [1, m])$, M_2 by $M([\frac{m}{2} + 1, m] \times [j, m])$, and M_3 by $M([1, \frac{m}{2} - 1] \times [1, j - 1])$. We define the *size* of a submatrix of M by the sum of its number of rows and number of columns, for example, M is of size $2m$, $|M_2| = \frac{3m}{2} - j + 1$, and $|M_3| = \frac{m}{2} + j - 2$.

Each recursive call is associated with a range of rows $[l, l']$ and a range of columns $[u', u]$ (the submatrix $M([l, l'] \times [u', u])$), and a corresponding input graph $G' = G([l, l'] \times [u', u])$. The scheme does not state which edges should be in G' or how to construct it, but it does require the followings properties:

1. The number of edges in G' should be $O(|M'|)$.
2. Given G', the feasibility decider can answer a feasibility query for any range in M', in $O(f(|G'|))$ time.
3. The construction of the graphs for the next level should take $O(|G'|)$ time.

The optimization scheme is given in Algorithm 1; its initial input is $G = G([1, m] \times [1, m])$.

Correctness. Let g be a bivariate real function with the property that for any four values of the weight function $c \leq a \leq b \leq d$, it holds that $g(a, b) \leq g(c, d)$. In our case, $g(a, b) = b - a$. We prove a somewhat more general theorem – that our scheme applies to any such monotone function g; for example, $g(a, b) = b/a$ (assuming the edge weights are positive numbers).

▶ **Theorem 8.** *Algorithm 1 returns the minimum value $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$.*

Proof. We claim that given a graph $G' = G([l, l'] \times [u', u])$ as input, Algorithm 1 returns the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [l, l']$ and $S_{max} \in [u', u]$. Let $M' = M([l, l'] \times [u', u])$ be the corresponding matrix. The proof is by induction on the number of rows in M'.

First, notice that the algorithm runs the feasibility decider only on ranges from M'. The base case is when M' contains a single row, i.e. $l = l'$. In this case the algorithm returns the minimal feasible range $[w_l, w_j]$ such that $j \in [u', u]$, or returns ∞ if there is no such range. Else, M' has more than one row. Assume that there is no feasible range in the middle row

of M'. In other words, there is no $j \in [u', u]$ such that $[w_i, w_j]$ is a feasible range. Trivially, for any $i' > i$ we have $w_{i'} > w_i$, and therefore for any $j \in [u', u]$, $[w_{i'}, w_j]$ is not a feasible range, and the algorithm continues recursively with $G_1 = G([l, i-1] \times [u', u])$. Now assume that $[w_i, w_j]$ is the minimal feasible range in the middle row. We can partition the ranges in M' to four types (submatrices):

1. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i+1, l']$ and $j' \in [j, u]$.
2. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i-1]$ and $j' \in [u', j-1]$.
3. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i, l']$ and $j' \in [u', j-1]$. For any such valid range ($j' > i'$), we have $[w_{i'}, w_{j'}] \subseteq [w_i, w_j]$, so it is not a feasible range (otherwise, the result of the binary search would be $[w_i, w_{j'}]$).
4. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i]$ and $j' \in [j, u]$. Since $j \geq i$, all these ranges are valid. For any such range, we have $w_{i'} \leq w_i \leq w_j \leq w_{j'}$, therefore, all these ranges are feasible, but since $g(w_i, w_j) \leq g(w_{i'}, w_{j'})$, there is no need to check them.

Indeed, the algorithm continues recursively with G_1 and G_2 (corresponding to ranges of type 1 and 2, respectively), which may contain smaller feasible ranges. By the induction hypothesis, the recursive calls return the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [i+1, l']$ and $S_{max} \in [j, u]$ or $S_{min} \in [l, i-1]$ and $S_{max} \in [u', j-1]$. Finally, the algorithm returns the minimum over all the feasible ranges in M'. ◀

▶ **Lemma 9.** *The total size of the matrices in each level of the recursion tree is at most* $2m$.

Proof. By induction on the level. The only matrix in level 0 is M, and $|M| = 2m$. Let $M' = M([l, l'] \times [u', u])$ be a matrix in level $i-1$. The size of M' is $l' - l + u - u' + 2$ (it has $l' - l + 1$ rows and $u - u' + 1$ columns). In level i we perform a binary search in the middle row of M' to find the smallest feasible range $[w_{\frac{l+l'}{2}}, w_j]$ in this row. It is easy to see that the resulting two submatrices are of sizes $l' - \frac{l+l'}{2} + u - j + 1$ and $\frac{l+l'}{2} - l + j - u'$, respectively, which sums to $l' - l + u - u' + 1$. ◀

Running time. Consider the recursion tree. It consists of $O(\log m)$ levels, where the i'th level is associated with 2^i disjoint submatrices of M. Level 0 is associated with the matrix $M_0 = M$, level 1 is associated with the submatrices M_2 and M_3 of M (see Figure 3), etc.

In the i'th level we apply Algorithm 1 to each of the 2^i submatrices associated with this level. Let $\{M_k^i\}_{k=1}^{2^i}$ be the submatrices associated with the i'th level. Let G_k^i be the graph corresponding to M_k^i. The size of G_k^i is linear in the size of M_k^i. The feasibility decider runs in $O(f(|M_k^i|))$ time, and thus the binary search in M_k^i runs in $O(f(|M_k^i|) \log |M_k^i|)$ time. Constructing the graphs for the next level takes $O(|M_k^i|)$ time. By lemma 9, the total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|) \log |M_k^i|)) \leq O(\sum_{k=1}^{2^i} |M_k^i| + \sum_{k=1}^{2^i} f(|M_k^i|) \log m) = O(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))$. Finally, the running time of the entire algorithm is $O(m \log m + \sum_{i=1}^{\log m}(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))) = O(m \log m + \log m \sum_{i=1}^{\log m} \sum_{k=1}^{2^i} f(|M_k^i|))$.

Notice that the number of potential ranges is $O(m^2)$, while the number of weights is only $O(m)$. Nevertheless, whenever $f(|M'|)$ is a linear function, our optimization scheme runs in $O(m \log^2 m)$ time. More generally, whenever $f(|M'|)$ is a function for which $f(x_1) + \cdots + f(x_k) = O(f(x_1 + \cdots + x_k))$, for any $x_1, \ldots, x_k$, our scheme runs in $O(m \log m + f(2m) \log^2 m)$ time.

5 MUPP and WDFD under translation in 1D

In Section 3 we described an algorithm for WDFD under translation in 1D, which uses a dynamic data structure due to Eppstein et al. [15]. In this section we present a much simpler algorithm for the problem, which avoids heavy tools and has roughly the same running time.

As shown in Section 3, WDFD under translation in 1D can be viewed as BOP. More precisely, we say that a range $[s,t]$ is a feasible range if (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$. Now, our goal is to find a feasible range of minimum size.

Consider the following weighted graph $\tilde{G}_w = (\tilde{V}_w, \tilde{E}_w, \omega)$, where $\tilde{V}_w = (A \times B) \cup \{v_e \mid e \in E_w\}$, $\tilde{E}_w = \{(u, v_e), (v_e, v) \mid e = (u, v) \in E_w\}$, and $\omega(((a_i, b_j), v_e)) = a_i - b_j$. In other words, $\tilde{G}_w$ is obtained from G_w by adding, for each edge $e = (u, v)$ of G_w, a new vertex v_e, which splits the edge into two new edges, $(u, v_e), (v_e, v)$, whose weight is the distance associated with their original vertex.

Now (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$, if and only if there exists a path P between (a_1, b_1) and (a_n, b_m) in G_w such that $\overline{V(P)} \in [s,t]$, if and only if there exists a path $\tilde{P}$ between (a_1, b_1) and (a_n, b_m) in $\tilde{G}_w$ such that for each edge $e \in \tilde{P}$, $\omega(e) \in [s,t]$.

We have reduced our problem to a special case of the most uniform path problem (MUPP). We show below how to apply our scheme to MUPP, with a linear-time feasibility decider, and thus obtain the following theorem as a by-product:

▶ **Theorem 10.** *Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points in 1D. Then, the weak discrete Fréchet distance under translation, $\widehat{d^w_{dF}}(A, B)$, can be computed in $O(mn \log^2(m+n))$ time.*

Most uniform path. Given a weighted graph $G = (V, E, w)$ with n vertices and m edges, and two vertices $s, t \in V$, the goal is to find a path P^* in G between s and t, which minimizes $\max\{w(e) : e \in P\} - \min\{w(e) : e \in P\}$, over all paths P between s and t.

Here $\mathcal{F}$ is the set of paths in G between s and t. The matrix for the initial call is M and G is its associated graph. Consider a recursive call, and let M' be the submatrix and G' the graph associated with it. Throughout the execution of the algorithm, we maintain the following properties: (i) The number of edges and vertices in G' is at most $O(|M'|)$, and (ii) Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.

Construction of the graphs for the next level. Given the input graph G' and a submatrix $M'' = M([p, p'] \times [q', q])$ of M', we construct the corresponding graph G'' as follows: First, we remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$. Then, we contract edges with weights in the range $(w_{p'}, w_{q'})$, and finally we remove all the isolated vertices. Notice that G'' is a graph minor of G', and, clearly, all the properties hold.

The feasibility decider. Let $[w_p, w_q]$ be a range from M'. Run a BFS in G', beginning from s, while ignoring edges with weights outside the range $[w_p, w_q]$. If the BFS finds t, return "yes", otherwise return "no". The algorithm returns "yes" if and only if there exists a path between s and t in G' with edges in the range $[w_p, w_q]$, i.e., if and only if such a path exists in G. The running time of the decider is $O(|G'|) = O(|M'|)$.

▶ **Theorem 11.** *The most uniform path problem in G can be solved in $O(m \log^2 n)$ time.*

▶ Remark. We have introduced an alternative optimization scheme for BOP and demonstrated its power. It would be interesting to find additional applications of this scheme. For example, using it we easily obtain an $O(m \log^2 n)$-time algorithm for the *Most Uniform Spanning Tree* problem; slower than the specialized algorithm of Galil and Schieber [17] by only a log-factor.

6 Discussion

In an unpublished manuscript [16], we suggested a new variant of DFD – the *discrete Fréchet gap*. Given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, the discrete Fréchet gap between them is the smallest range $[s, t]$, $s \geq t \geq 0$, for which (a_n, b_m) is a reachable position w.r.t. $\sigma_{[s,t]}$, where $\sigma_{[s,t]}(a_i, b_j) = 1$ if and only if $d(a_i, b_j) \in [s, t]$. We used a less general version of our scheme for BOP to solve two variants of the gap problem: the *discrete Fréchet gap with shortcuts* (where (a_n, b_m) is an s-reachable position), and the *weak discrete Fréchet gap* (where (a_n, b_m) is a w-reachable position).

It is interesting to note that DFDS and WDFD, both in 1D under translation, are in some sense analogous to their respective gap variants (in d dimensions and no translation): We can use similar algorithms to compute them, but with different indicator functions. This connection supports the intuition that there is some connection between the discrete Fréchet gap and DFD under translation.

References

1 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014. doi:10.1137/130920526.

2 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the fréchet distance. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2001. doi:10.1007/3-540-44693-1_6.

4 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete fréchet distance with shortcuts via approximate distance counting and selection. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 377. ACM, 2014. doi:10.1145/2582112.2582155.

5 Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. A faster algorithm for the discrete fréchet distance under translation. *CoRR*, abs/1501.03724, 2015. arXiv:1501.03724.

6 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 853–864. ACM, 2005. URL: http://www.vldb.org/archives/website/2005/program/paper/fri/p853-brakatsoulas.pdf.

7 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

8 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog - with an application to alt's conjecture. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1399–1413. SIAM, 2014. doi:10.1137/1.9781611973402.103.

9 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the fréchet distance. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 645–654. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496841.

10 Maike Buchin. *On the computability of the Fréchet distance between triangulated surfaces*. PhD thesis, FU Berlin, 2007.

11 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the fréchet distance with shortcuts is np-hard. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 367. ACM, 2014. doi:10.1145/2582112.2582144.

12 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

13 Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *Journal of Mathematical Imaging and Vision*, 27(3):203–216, 2007. doi:10.1007/s10851-006-0647-0.

14 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Information Systems Dept., Technical University of Vienna, 1994.

15 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. doi:10.1016/0196-6774(92)90004-V.

16 Omrit Filtser and Matthew J. Katz. The discrete fréchet gap. *CoRR*, abs/1506.04861, 2015. arXiv:1506.04861.

17 Zvi Galil and Baruch Schieber. On finding most uniform spanning trees. *Discrete Applied Mathematics*, 20(2):173–175, 1988. doi:10.1016/0166-218X(88)90062-5.

18 Pierre Hansen, Giovanni Storchi, and Tsevi Vovor. Paths with minimum range and ratio of arc lengths. *Discrete Applied Mathematics*, 78(1-3):89–102, 1997. doi:10.1016/S0166-218X(97)00008-5.

19 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

20 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete fréchet distance. *J. Bioinformatics and Computational Biology*, 6(1):51–64, 2008. doi:10.1142/S0219720008003278.

21 Silvano Martello, WR Pulleyblank, Paolo Toth, and Dominique De Werra. Balanced optimization problems. *Operations Research Letters*, 3(5):275–278, 1984.

22 Axel Mosig and Michael Clausen. Approximately matching polygonal curves with respect to the fre'chet distance. *Comput. Geom.*, 30(2):113–127, 2005. doi:10.1016/j.comgeo.2004.05.004.

23 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

24 Carola Wenk. *Shape matching in higher dimensions*. PhD thesis, Free University of Berlin, Dahlem, Germany, 2003. URL: http://www.diss.fu-berlin.de/2003/151/index.html.

Partial Complementation of Graphs

Fedor V. Fomin[1]
Department of Informatics, University of Bergen, Norway
fedor.fomin@ii.uib.no
https://orcid.org/0000-0003-1955-4612

Petr A. Golovach[2]
Department of Informatics, University of Bergen, Norway
petr.golovach@ii.uib.no
https://orcid.org/0000-0002-2619-2990

Torstein J. F. Strømme[3]
Department of Informatics, University of Bergen, Norway
torstein.stromme@ii.uib.no
https://orcid.org/0000-0002-3896-3166

Dimitrios M. Thilikos[4]
AlGCo project-team, LIRMM, Université de Montpellier, CNRS, France.
Department of Mathematics National and Kapodistrian University of Athens, Greece
sedthilk@thilikos.info
https://orcid.org/0000-0003-0470-1800

Abstract

A *partial complement* of the graph G is a graph obtained from G by complementing all the edges in one of its induced subgraphs. We study the following algorithmic question: for a given graph G and graph class $\mathcal{G}$, is there a partial complement of G which is in $\mathcal{G}$? We show that this problem can be solved in polynomial time for various choices of the graphs class $\mathcal{G}$, such as bipartite, degenerate, or cographs. We complement these results by proving that the problem is NP-complete when $\mathcal{G}$ is the class of r-regular graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis

Keywords and phrases Partial complementation, graph editing, graph classes

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.21

Related Version A full version of the paper is available at `http://arxiv.org/abs/1804.10920`

Acknowledgements We thank Saket Saurabh for helpful discussions, and also a great thanks to the anonymous reviewers who provided valuable feedback.

1 Introduction

One of the most important questions in graph theory concerns the efficiency of recognition of a graph class $\mathcal{G}$. For example, how fast we can decide whether a graph is chordal, 2-connected,

[1] Supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL".
[2] Supported by the Research Council of Norway via the project "CLASSIS".
[3] Supported by the Research Council of Norway via the project "MULTIVAL".
[4] Supported by project "DEMOGRAPH" (ANR-16-CE40-0028).

triangle-free, of bounded treewidth, bipartite, 3-colorable, or excludes some fixed graph as a minor? In particular, the recent developments in parameterized algorithms are driven by the problems of recognizing of graph classes which do not differ up to a "small disturbance" from graph classes recognizable in polynomial time. The amount of disturbance is quantified in "atomic" operations required for modifying an input graph into the "well-behaving" graph class $\mathcal{G}$. The standard operations could be edge/vertex deletions, additions or edge contractions. Many problems in graph algorithms fall into this graph modification category: is it possible to add at most k edges to make a graph 2-edge connected or to make it chordal? Or is it possible to delete at most k vertices such that the resulting graph has no edges or contains no cycles?

A rich subclass of modification problems concerns edge editing problems. Here the "atomic" operation is the change of adjacency, i. e. for a pair of vertices u, v, we can either add an edge uv or delete the edge uv. For example, the Cluster Editing problem asks to transform an input graph into a cluster graph, that is a disjoint union of cliques, by flipping at most k adjacency relations.

Besides the basic edge editing, it is natural to consider problems where the set of removed and added edges should satisfy some structural constraints. In particular, such problems were considered for *complementation* problems. Recall that the *complement* of a graph G is a graph H on the same vertices such that two distinct vertices of H are adjacent if and only if they are not adjacent in G. Seidel (see [19, 20, 21]) introduced the operation that is now known as the *Seidel switch*. For a vertex v of a graph G, this operation complements the adjacencies of v, that is, it removes the edges incident to v and makes v adjacent to the non-neighbors of v in G. Respectively, for a set of vertices U, the Seidel switching, that is, the consecutive switching for the vertices of U, complements the adjacencies between U and its complement $V(G) \setminus U$. The study of the algorithmic question whether it is possible to obtain a graph from a given graph class by the Seidel switch was initiated by Ehrenfeucht et al. [7]. Further results were established in [11, 12, 13, 16, 15]. Another important operation of this type is the *local complementation*. For a vertex v of a graph G, the *local complementation of G at v* is the graph obtained from G by replacing $G[N(v)]$ by its complement. This operation plays crucial role in the definition of *vertex-minors* [17] and was investigated in this contest (see, e.g. [6, 18]). See also [2, 14] for some algorithmic results concerning local complementations.

In this paper we study the *partial complement* of a graph, which was introduced by Kamiński, Lozin, and Milanič in [14] in their study of the clique-width of a graph. A *partial complement* of a graph G is a graph obtained from G by complementing all the edges of one of its induced subgraphs. More formally, for a graph G and $S \subseteq V(G)$, we define $G \oplus S$ as the graph with the vertex set $V(G)$ whose edge set is defined as follows: a pair of distinct vertices u, v is an edge of $G \oplus S$ if and only if one of the following holds:

- $uv \in E(G) \wedge (u \notin S \vee v \notin S)$, or
- $uv \notin E(G) \wedge u \in S \wedge v \in S$.

Thus when the set S consists only of two vertices $\{u, v\}$, then the operation changes the adjacency between u and v, and for a larger set S, $G \oplus S$ changes the adjacency relations for all pairs of vertices of S.

We say that a graph H is a partial complement of the graph G if H is isomorphic to $G \oplus S$ for some $S \subseteq V(G)$. For a graph class $\mathcal{G}$ and a graph G, we say that there is a *partial complement of G to $\mathcal{G}$* if for some $S \subseteq V(G)$, we have $G \oplus S \in \mathcal{G}$. We denote by $\mathcal{G}^{(1)}$ the class of graphs such that its members can be partially complemented to $\mathcal{G}$.

Let $\mathcal{G}$ be a graph class. We consider the following generic algorithmic problem.

Partial Complement to $\mathcal{G}$ (PC$\mathcal{G}$)
Input: A simple undirected graph G.
Question: Is there a partial complement of G to $\mathcal{G}$?

In other words, how difficult is it to recognize the class $\mathcal{G}^{(1)}$? In this paper we show that there are many well-known graph classes $\mathcal{G}$ such that $\mathcal{G}^{(1)}$ is recognizable in polynomial time. We show that

- Partial Complement to $\mathcal{G}$ is solvable in time $\mathcal{O}(f(n)\cdot n^4+n^6)$ when $\mathcal{G}$ is a triangle-free graph class recognizable in time $f(n)$. For example, this implies that when $\mathcal{G}$ is the class of bipartite graphs, the class $\mathcal{G}^{(1)}$ is recognizable in polynomial time.
- Partial Complement to $\mathcal{G}$ is solvable in time $f(n)\cdot n^{\mathcal{O}(1)}$ when $\mathcal{G}$ is a d-degenerate graph class recognizable in time $f(n)$. Thus when $\mathcal{G}$ is the class of planar graphs, class of cubic graphs, class of graph of bounded treewidth, or class of H-minor free graphs, then the class $\mathcal{G}^{(1)}$ is recognizable in polynomial time.
- Partial Complement to $\mathcal{G}$ is solvable in polynomial time when $\mathcal{G}$ is a class of bounded clique-width expressible in monadic second-order logic (with no edge set quantification). In particular, if $\mathcal{G}$ is the class of P_4-free graphs (cographs), then $\mathcal{G}^{(1)}$ is recognizable in polynomial time.
- Partial Complement to $\mathcal{G}$ is solvable in polynomial time when $\mathcal{G}$ can be described by a 2×2 M-partition matrix. Therefore $\mathcal{G}^{(1)}$ is recognizable in polynomial time when $\mathcal{G}$ is the class of split graphs, as they can be described by such a matrix.

Nevertheless, there are cases when the problem is NP-hard. In particular, we prove that this holds when $\mathcal{G}$ is the class of r-regular graphs.

2 Partial complementation to triangle-free graph classes

A triangle is a complete graph on three vertices. Many graph classes does not allow the triangle as a subgraph, for instance trees, forests, or graphs with large girth. In this paper we show that partial complementation to triangle-free graphs can be decided in polynomial time.

More precisely, we show that if a graph class $\mathcal{G}$ can be recognized in polynomial time and it is triangle-free, then we can also solve Partial Complement to $\mathcal{G}$ in polynomial time.

Our algorithm is constructive, and returns a *solution* $S\subseteq V(G)$, that is a set S such that $G\oplus S$ is in $\mathcal{G}$. We say that a solution *hits* an edge uv (or a non-edge $\overline{uv}$), if both u and v are contained in S.

Our algorithm considers each of the following cases.

- **(i)** There is a solution S of size at most two.
- **(ii)** There is a solution S containing two vertices that are non-adjacent in G.
- **(iii)** There is a solution S such that it form a clique of size at least 3 in G.
- **(iv)** G is a no-instance.

Case (i) can be resolved in polynomial time by brute-force, and thus we start from analyzing the structure of a solution in Case (ii). We need the following observation.

▶ **Observation 1.** *Let $\mathcal{G}$ be a class of triangle-free graphs and let G be an instance of* Partial Complement to $\mathcal{G}$*, where $S\subseteq V(G)$ is a valid solution. Then*

- **a)** *$G[S]$ does not contain an independent set of size 3, and*
- **b)** *for every triangle $\{u,v,w\}\subseteq V(G)$, at least two vertices are in S.*

Because all non-edges between vertices in $G[S]$ become edges in $G \oplus S$ and vice versa, whereas all (non-) edges with an endpoint outside S remain untouched, we see that the observation holds.

Let us recall that a graph G is a *split graph* if its vertex set can be partitioned into $V(G) = C \cup I$, where C is a clique and I is an independent set. Let us note that the vertex set of a split graph can have several *split partitions*, i.e. partitions into a clique and independent set. However, the number of split partitions of an n-vertex split graphs is at most n. The analysis of Case (ii) is based on the following lemma.

▶ **Lemma 2.** *Let $\mathcal{G}$ be a class of triangle-free graphs and let G be an instance of* Partial Complement to $\mathcal{G}$*. Let $S \subseteq V(G)$ be a valid solution which is not a clique, and let $u, v \in S$ be distinct vertices such that $uv \notin E(G)$. Then*

a) *the entire solution S is a subset of the union of the closed neighborhoods of u and v, that is $S \subseteq N_G[u] \cup N_G[v]$;*

b) *every common neighbor of u and v must be contained in the solution S, that is $N_G(u) \cap N_G(v) \subseteq S$;*

c) *the graph $G[N(u) \setminus N(v)]$ is a split graph. Moreover, $(N(u) \setminus N(v)) \cap S$ is a clique and $(N(u) \setminus N(v)) \setminus S$ is an independent set.*

Proof. We will prove each point separately, and in order.

a) Assume for the sake of contradiction that the solution S contains a vertex $w \notin N_G[u] \cup N_G[v]$. But then $\{u, v, w\}$ is an independent set in G, which contradicts item a) of Observation 1.

b) Assume for the sake of contradiction that the solution S does not contain a vertex $w \in N_G(u) \cap N_G(v)$. Then the edges uw and vw will both be present in $G \oplus S$, as well as the edge uv. Together, these forms a triangle.

c) We first claim that the solution S is a vertex cover for $G[N(u) \setminus N(v)]$. If it was not, then there would exist an edge u_1u_2 of $G[N(u) \setminus N(v)]$ such that both endpoints $u_1, u_2 \notin S$, yet u_1, u_2 would form a triangle with u in $G \oplus S$, which would be a contradiction. Hence $(N(u) \setminus N(v)) \setminus S$ is an independent set. Secondly, we claim that $(N(u) \setminus N(v)) \cap S$ forms a clique. If not, then there would exist $u_1, u_2 \in (N(u) \setminus N(v)) \cap S$ which are nonadjacent. In this case $\{u_1, u_2, v\}$ is an independent set, which contradicts item a) of Observation 1. Taken together, these claims imply the last item of the lemma. ◀

We now move on to examine the structure of a solution for the third case, when there exists a solution which is a clique of size at least three.

▶ **Lemma 3.** *Let $\mathcal{G}$ be a class of triangle-free graphs and let G be an instance of* Partial Complement to $\mathcal{G}$*. Let $S \subseteq V(G)$ be a solution such that $|S| \geq 3$ and $G[S]$ is a clique. Let $u, v \in S$ be distinct. Then*

a) *the solution S is contained in their common neighborhood, that is $S \subseteq N_G[u] \cap N_G[v]$, and*

b) *the graph $G[N_G[u] \cap N_G[v]]$ is a split graph where $(N_G[u] \cap N_G[v]) \setminus S$ is an independent set.*

Proof. We prove each point separately, and in order.

a) Assume for the sake of contradiction that the solution S contains a vertex w which is not in the neighborhood of both u and v. This contradicts that S is a clique.

b) We claim that S is a vertex cover of $G[N_G[u] \cap N_G[v]]$. Because S is also a clique, the statement of the lemma will then follow immediately. Assume for the sake of contradiction that S is not a vertex cover. Then there exist an uncovered edge w_1w_2, where $w_1, w_2 \in N_G[u] \cap N_G[v]$, and also $w_1, w_2 \notin S$. Since $\{u, w_1, w_2\}$ form a triangle,

we have by b) of Observation 1 that at least two of these vertices are in S. That is a contradiction, so our claim holds. ◀

We now have everything in place to present the algorithm.

▶ **Algorithm 4** (Partial Complement to $\mathcal{G}$ where $\mathcal{G}$ is triangle-free).

Input: An instance G of PC$\mathcal{G}$ where $\mathcal{G}$ is a triangle-free graph class recognizable in time $f(n)$ for some function f.

Output: A set $S \subseteq V(G)$ such that $G \oplus S$ is in $\mathcal{G}$, or a correct report that no such set exists.

1. By brute force, check if there is a solution of size at most 2. If yes, return this solution.
2. For every non-edge $\overline{uv}$ of G:
   a. If either $G[N(u) \setminus N_G(v)]$ or $G[N_G(u) \setminus N_G(v)]$ is not a split graph, skip this iteration and try the next non-edge.
   b. Let (I_u, C_u) and (I_v, C_v) denote a split partition of $G[N_G(u) \setminus N_G(v)]$ and $G[N_G(v) \setminus N_G(u)]$ respectively. For each pair of split partitions $(I_u, C_u), (I_v, C_v)$:
      i. Construct solution candidate $S' := \{u, v\} \cup (N_G(u) \cap N_G(v)) \cup C_u \cup C_v$
      ii. If $G \oplus S'$ is a member of $\mathcal{G}$, return S'
3. Find a triangle $\{x, y, z\}$ of G
4. For each edge in the triangle $uv \in \{xy, xz, yz\}$:
   a. If $G[N_G(u) \cap N_G(v)]$ is not a split graph, skip this iteration and try the next edge.
   b. For each possible split partition (I, C) of $G[N_G(u) \cap N_G(v)]$:
      i. Construct solution candidate $S' := \{u, v\} \cup C$
      ii. If $G \oplus S'$ is a member of $\mathcal{G}$, return S'
5. Return 'None'

▶ **Theorem 5.** *Let $\mathcal{G}$ be a class of triangle-free graphs such that deciding whether an n-vertex graph is in $\mathcal{G}$ is solvable in time $f(n)$ for some function f. Then* Partial Complement to $\mathcal{G}$ *is solvable in time $\mathcal{O}(n^6 + n^4 \cdot f(n))$.*

Proof. We will prove that Algorithm 4 is correct, and that its running time is $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$. We begin by proving correctness. Step 1 is trivially correct. After Step 1 we can assume that any valid solution has size at least three, and we have handled Case (i) when there exists a solution of size at most two. We have the three cases left to consider: (ii) There exists a solution which hits a non-edge, (iii) there is a solution S such that in $G \oplus S$ vertices of S form a clique of size at least 3, and (iv) no solution exists.

In the case that there exists a solution S hitting a non-edge uv, we will at some point guess this non-edge in Step 2 of the algorithm. By Lemma 2, we have that both $G[N_G(u) \setminus N_G(v)]$ and $G[N_G(u) \setminus N_G(v)]$ are split graphs, so we do not miss the solution S in Step 2a. Since we try every possible combinations of split partitions in Step 2b, we will by Lemma 2 at some point construct S' correctly such that $S' = S$.

In the case that there exist only solutions which hits exactly a clique, we first find some triangle $\{x, y, z\}$ of G. It must exist, since a solution S is a clique of size at least three. By Observation 1b, at least two vertices of the triangle must be in the S. At some point in step 4 we guess these vertices correctly. By Lemma 3b we know that $G[N_G(u) \cap N_G(v)]$ is a split graph, so we will not miss S in Step 4a. Since we try every split partition in Step 4b, we will by Lemma 3 at some point construct S' correctly such that $S' = S$.

Lastly, in the case that there is no solution, we know that there neither exists a solution of size at most two, nor a solution which hits a non-edge, nor a solution which hits a clique of size at least three. Since these three cases exhaust the possibilities, we can correctly report that there is no solution when none was found in the previous steps.

For the runtime, we start by observing that Step 1 takes time $\mathcal{O}(n^2 \cdot f(n))$. The sub-procedure of Step 2 is performed $\mathcal{O}(n^2)$ times, where step 2a takes time $\mathcal{O}(n \log n)$. The sub-procedure of Step 2b takes time at most $\mathcal{O}(n^2 + f(n))$, and it is performed at most $\mathcal{O}(n^2)$ times. In total, Step 2 will use no longer than $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$ time. Step 3 is trivially done in time $\mathcal{O}(n^3)$. The sub-procedure of Step 4 is performed at most three times. Step 4a is done in $\mathcal{O}(n \log n)$ time, and step 4b is done in $\mathcal{O}(n \cdot (n^2 + f(n))$ time, which also becomes the asymptotic runtime of the entire step 4. The worst running time among these steps is Step 2, and as such the runtime of Algorithm 4 is $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$. ◀

3 Complement to degenerate graphs

For $d > 0$, we say that a graph G is d-degenerate, if every induced (not necessarily proper) subgraph of G has a vertex of degree at most d. For example, trees are 1-degenerate, while planar graphs are 5-degenerate.

▶ **Theorem 6.** *Let $\mathcal{G}$ be a class of d-degenerate graphs such that deciding whether an n-vertex graph is in $\mathcal{G}$ is solvable in time $f(n)$ for some function f. Then* Partial Complement to $\mathcal{G}$ *is solvable in time $f(n) \cdot n^{2^{\mathcal{O}(d)}}$.*

Proof. Let G be an n-vertex graph. We are looking for a vertex subset S of G such that $G \oplus S \in \mathcal{G}$.

We start from trying all vertex subsets of G of size at most $2d$ as a candidate for S. Thus, in time $\mathcal{O}(n^{2d} \cdot f(n))$ we either find a solution or conclude that a solution, if it exists, should be of size more than $2d$.

Now we assume that $|S| > 2d$. We try all subsets of $V(G)$ of size $2d + 1$. Then if G can be complemented to $\mathcal{G}$, at least one of these sets, say X, is a subset of S. In total, we enumerate $\binom{n}{2d+1}$ sets.

First we consider the set Y of all vertices in $V(G) \setminus X$ with at least $d + 1$ neighbors in X. The observation here is that most vertices from Y are in S. More precisely, if more than

$$\alpha = \binom{|X|}{d+1} \cdot d + 1 = \binom{2d+1}{d+1} \cdot d + 1$$

vertices of Y are not in S, then $G \oplus S$ contains a complete bipartite graph $G_{d+1,d+1}$ as a subgraph, and hence $G \oplus S$ is not d-degenerate. Thus, we make at most $\binom{n}{\alpha}$ guesses on which subset of Y is in S.

Similarly, when we consider the set Z of all vertices from $V(G) \setminus X$ with at most d neighbors in X, we have that at most α of vertices from Z could belong to S. Since $V(G) = X \cup Y \cup Z$, if there is a solution S, it will be found in at least one from

$$\binom{n}{2d+1} \cdot \alpha^2 = n^{2^{\mathcal{O}(d)}}$$

of the guesses. Since for each set S we can check in time $f(n)$ whether $G \oplus S \in \mathcal{G}$, this concludes the proof. ◀

4 Complement to M-partition

Many graph classes can be defined by whether it is possible to partition the vertices of graphs in the class such that certain internal and external edge requirements of the parts are met. For instance, a complete bipartite graph is one which can be partitioned into two sets such

that every edge between the two sets is present (external requirement), and no edge exists within any of the partitions (internal requirements). Other examples are split graphs and k-colorable graphs. Feder et al. [8] formalized such partition properties of graph classes by making use of a symmetric matrix over $\{0, 1, \star\}$, called an *M-partition*.

▶ **Definition 7** (M-partition). For a $k \times k$ matrix M, we say that a graph G belongs to the graph class $\mathcal{G}_M$ if its vertices can be partitioned into k (possibly empty) sets $X_1, X_2, \ldots, X_k$ such that, for every $i \in [k]$, if

- $M[i,i] = 1$, then X_i is a clique and if $M[i,i] = 0$, then X_i is an independent set, and

for every $i, j \in [k]$, $i \neq j$,

- if $M[i,j] = 1$, then every vertex of X_i is adjacent to all vertices of X_j,
- if $M[i,j] = 0$, then there is no edges between X_i and X_j.

Note that if $M[i,j] = \star$, then there is no restriction on the edges between vertices from X_i and X_j.

For example, for matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 0 \end{pmatrix}$$

the corresponding class of graphs is the class of bipartite graphs, while matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 1 \end{pmatrix}$$

identifies the class of split graphs.

In this section we prove the following theorem.

▶ **Theorem 8.** *Let $\mathcal{G} = \mathcal{G}_M$ be a graph class described by an M-partition matrix of size 2×2. Then* Partial Complement to $\mathcal{G}$ *is solvable in polynomial time.*

In particular, Theorem 8 yields polynomial algorithms for Partial Complement to $\mathcal{G}$ when $\mathcal{G}$ is the class of split graphs or (complete) bipartite graphs. The proof of our theorem is based on the following beautiful dichotomy result of Feder et al. [8] on the recognition of classes $\mathcal{G}_M$ described by 4×4 matrices.

▶ **Proposition 9** ([8, Corollary 6.3]). *Suppose M is a symmetric matrix over $\{0, 1, \star\}$ of size $k = 4$. Then the recognition problem for $\mathcal{G}_M$ is*

- *NP-complete when M contains the matrix for 3-coloring or its complement, and no diagonal entry is $\star$.*
- *Polynomial time solvable otherwise.*

▶ **Lemma 10.** *Let M be a symmetric $k \times k$ matrix giving rise to the graph class $\mathcal{G}_M = \mathcal{G}$. Then there exists a $2k \times 2k$ matrix M' such that for any input G to* Partial Complement to $\mathcal{G}$*, it is a yes-instance if and only if G belongs to $\mathcal{G}_{M'}$.*

Proof. Given M, we construct a matrix M' in linear time. We let M' be a matrix of dimension $2k \times 2k$, where entry $M'[i,j]$ is defined as $M[\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil]$ if at least one of i, j is even, and $\neg M[\frac{i+1}{2}, \frac{j+1}{2}]$ if i, j are both odd. Here, $\neg 1 = 0$, $\neg 0 = 1$, and $\neg \star = \star$. For example, for matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 1 \end{pmatrix}$$

the above construction results in

$$M' = \begin{pmatrix} 1 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ \star & \star & 0 & 1 \\ \star & \star & 1 & 1 \end{pmatrix}.$$

We prove the two directions separately.

($\Longrightarrow$) Assume there is a partial complementation $G \oplus S$ into $\mathcal{G}_M$. Let $X_1, X_2, \ldots, X_k$ be an M-partition of $G \oplus S$. We define partition $X'_1, X'_2, \ldots, X'_{2k}$ of G as follows. For every vertex $v \in X_i$, $1 \leq i \leq k$, we assign v to X'_{2i-1} if $v \in S$ and to X'_{2i} otherwise.

We now show that every edge of G respects the requirements of M'. Let $uv \in E(G)$ be an edge, and let $u \in X_i$ and $v \in X_j$. If at least one vertex from $\{u, v\}$, say v is not in S, then uv is also an edge in $G \oplus S$, thus $M[i, j] \neq 0$. Since $v \notin S$, it belongs to set $v \in X'_{2j}$. Vertex u is assigned to set X'_ℓ, where ℓ is either $2i$ or $2i - 1$, depending whether u belongs to S or not. But because $2j$ is even irrespectively of ℓ, $M'[\ell, 2j] = M[i, j] \neq 0$.

Now consider the case when both $u, v \in S$. Then the edge does not persist after the partial complementation by S, and thus $M[i, j] \neq 1$. We further know that u is assigned to X'_{2i-1} and v to X'_{2j-1}. Both $2i - 1$ and $2j - 1$ are odd, and by the construction of M', we have that $M'[2i - 1, 2j - 1] \neq 0$, and again the edge uv respects M'. An analogous argument shows that also all non-edges respect M'.

($\Longleftarrow$) Assume that there is a partition $X'_1, X'_2, \ldots, X'_{2k}$ of G according to M'. Let the set S consist of all vertices in odd-indexed parts of the partition. We now show that $G \oplus S$ can be partitioned according to M. We define partition $X_1, X_2, \ldots, X_k$ by assigning each vertex $u \in X'_i$ to $X_{\lceil \frac{i}{2} \rceil}$. It remains to show that $X_1, X_2, \ldots, X_k$ is an M-partition of $G \oplus S$.

Let $u \in X_i$, $v \in X_j$. Suppose first that $uv \in E(G \oplus S)$. If at least one of u, v is not in S, we assume without loss of generality that $v \notin S$. Then $uv \in E(G)$ and $v \in X'_{2j}$. For vertex $u \in X'_\ell$, irrespectively, whether ℓ is $2i$ or $2i - 1$, we have that $M'[\ell, 2j] = M[i, j] \neq 0$. But then $M[i, j] \neq 0$. Otherwise we have $u, v \in S$. Then uv is a non-edge in G, and thus $M'[2i - 1, 2j - 1] \neq 1$. But by the construction of M', we have that $M[i, j] \neq 0$, and there is no violation of M. An analogous argument shows that if u and v are not adjacent in $G \oplus S$, it holds that $M[i, j] \neq 1$. Thus $X_1, X_2, \ldots, X_k$ is an M-partition of $G \oplus S$, which concludes the proof. ◀

Now we are ready to prove Theorem 8.

Proof of Theorem 8. For a given matrix M, we use Lemma 10 to construct a matrix M'. Let us note that by the construction of matrix M', for every 2×2 matrix M we have that matrix M' has at most two 1's and at most two 0's along the diagonal. Then by Proposition 9, the recognition of whether G admits M'-partition is in P. Thus by Lemma 10, Partial Complement to $\mathcal{G}$ is solvable in polynomial time ◀

5 Partial complementation to graph classes of bounded clique-width

We show that Partial Complement to $\mathcal{G}$ can be solved in polynomial time when $\mathcal{G}$ has bounded clique-width and can be expressed by an $\mathbf{MSO}_1$ property. We refer to the book [3] for the basic definitions. We will use the following result of Hliněný and Oum [10].

▶ **Proposition 11** ([10]). *There is an algorithm that for every integer k and graph G in time $O(|V(G)|^3)$ either computes a $(2^{k+1} - 1)$ expression for a graph G or correctly concludes that the clique-width of G is more than k.*

Note that the algorithm of Hliněný and Oum only approximates the clique-width but does not provide an algorithm to construct an optimal k-expression tree for a graph G of clique-width at most k. But this approximation is usually sufficient for algorithmic purposes.

Courcelle, Makowsky and Rotics [4] proved that every graph property that can be expressed in $\mathbf{MSO}_1$ can be recognized in linear time for graphs of bounded clique-width when given a k-expression.

▶ **Proposition 12** ([4, Theorem 4]). *Let $\mathcal{G}$ be some class of graphs of clique-width at most k such that for each graph $G \in \mathcal{G}$, a corresponding k-expression can be found in $\mathcal{O}(f(n,m))$ time. Then every $\boldsymbol{MSO}_1$ property on $\mathcal{G}$ can be recognized in time $\mathcal{O}(f(n,m)+n)$.*

The nice property of graphs with bounded clique-width is that their partial complementation is also bounded. In particular, Kamiński, Lozin, and Milanič in [14] observed that if G is a graph of clique-width k, then any partial complementation of G is of clique-width at most $g(k)$ for some computable function g. For completeness, we provide a more accurate upper bound whose proof is omitted in this extended abstract.

▶ **Lemma 13.** *Let G be a graph, $S \subseteq V(G)$. Then* $\textsc{cwd}(G \oplus S) \leq 3\textsc{cwd}(G)$.

▶ **Lemma 14.** *Let φ be an $\boldsymbol{MSO}_1$ property describing the graph class $\mathcal{G}$. Then there exists an $\boldsymbol{MSO}_1$ property ϕ describing the graph class $\mathcal{G}^{(1)}$ of size $|\phi| \in \mathcal{O}(|\varphi|)$.*

Proof. We will construct ϕ from φ in the following way: We start by prepending $\exists S \subseteq V(G)$. Then for each assessment of the existence of an edge in φ, say $uv \in E(G)$, replace that term with $((u \notin S \vee v \notin S) \wedge uv \in E(G)) \vee (u \in S \wedge v \in S \wedge uv \notin E(G))$. Symmetrically, for each assessment of the non-existence of an edge $uv \notin E(G)$, replace that term with $((u \notin S \vee v \notin S) \wedge uv \notin E(G)) \vee (u \in S \wedge v \in S \wedge uv \in E(G))$.

We observe that if φ is satisfiable for some graph G, then for every $S \subseteq V(G)$, the partial complementation $G \oplus S$ will yield a satisfying assignment to ϕ. Conversely, if ϕ is satisfiable for a graph G, then there exist some S such that φ is satisfied for $G \oplus S$. For the size, we note that each existence check for edges blows up by a constant factor. ◀

We are ready to prove the main result of this section.

▶ **Theorem 15.** *Let $\mathcal{G}$ be a graph class expressible in $\boldsymbol{MSO}_1$ which has bounded clique-width. Then* Partial Complement to $\mathcal{G}$ *is solvable in polynomial time.*

Proof. Let φ be the $\mathbf{MSO}_1$ formula which describes $\mathcal{G}$, and let G be an n-vertex input graph. We apply Proposition 11 for G and in time $O(n^3)$ either obtain a $(2^{3k+1}-1)$ expression for G or conclude that the clique-width of G is more than $3k$. In the latter case, by Lemma 13, G cannot be partially complemented to $\mathcal{G}$.

We then obtain an $\mathbf{MSO}_1$ formula ϕ from Lemma 14, and apply Proposition 12, which works in time $f(k,\phi) \cdot n$ for some function f. In total, the runtime of the algorithm is $f(k,\phi) \cdot n + n^3$. ◀

We remark that if clique-width expression is provided along with the input graphs, and $\mathcal{G}$ can be expressed in $\mathbf{MSO}_1$, then there is a linear time algorithm for Partial Complement to $\mathcal{G}$. This follows directly from Lemma 14 and Proposition 12.

Theorem 15 implies that for every class of graphs $\mathcal{G}$ of bounded clique-width characterized by a finite set of finite forbidden induced subgraphs, e. g. P_4-free graphs (also known as cographs) or classes of graphs discussed in [1], the Partial Complement to $\mathcal{G}$ problem is solvable in polynomial time. However, Theorem 15 does not imply that Partial Complement to $\mathcal{G}$ is solvable in polynomial time for $\mathcal{G}$ being of the class of graphs having

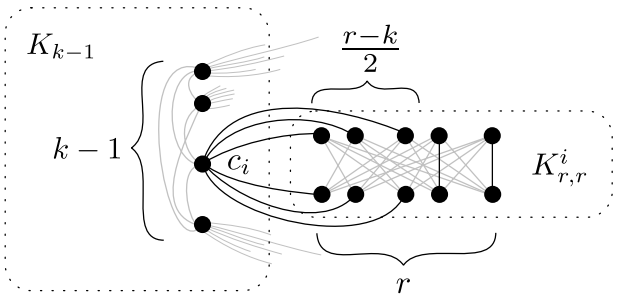


Figure 1 The graph $\text{GDG}_{k,r}$ is built of k parts, namely a clique K_{k-1}, and $k-1$ complete bipartite graphs $K^1_{r,r}, \ldots, K^{k-1}_{r,r}$ with some rewiring.

clique-width at most k. This is because such a class $\mathcal{G}$ cannot be described by $\mathbf{MSO}_1$. Interestingly, for the related class $\mathcal{G}$ of graphs of bounded *rank-width* (see [5] for the definition) at most k, the result of Oum and Courcelle [6] combined with Theorem 15 implies that Partial Complement to $\mathcal{G}$ is solvable in polynomial time.

6 Hardness of partial complementation to r-regular graphs

Let us remind that a graph G is r-regular if all its vertices are of degree r. We consider the following restricted version of Partial Complement to $\mathcal{G}$.

Partial Complement to r-Regular (PCrR)
Input: A simple undirected graph G, a positive integer r.
Question: Does there exist a vertex set $S \subseteq V(G)$ such that $G \oplus S$ is r-regular?

In this section, we show that Partial Complement to r-Regular is NP-complete by a reduction from Clique in r-regular Graph.

Clique in r-regular Graph (KrR)
Input: A simple undirected graph G which is r-regular, a positive integer k.
Question: Does G contain a clique on k vertices?

We will need the following well-known proposition.

▶ **Proposition 16** ([9]). Clique in r-regular Graph *is NP-complete.*

▶ **Theorem 17.** Partial Complement to r-Regular *is NP-complete.*

Proof. We begin by defining a gadget which we will use in the reduction. For integers $r > k$ such that $r - k$ is even, we build the graph $\text{GDG}_{k,r}$ as follows. Initially, we let $\text{GDG}_{k,r}$ consist of one clique on $k-1$ vertices, as well as $k-1$ distinct copies of $K_{r,r}$. These are all the vertices of the gadget, which is a total of $(k-1) + 2r \cdot (k-1)$ vertices. We denote the vertices of the clique $c_1, c_2, \ldots, c_{k-1}$, and we let the complete bipartite graphs be denoted by $K^1_{r,r}, K^2_{r,r}, \ldots, K^{k-1}_{r,r}$. For a bipartite graph $K^i_{r,r}$, let the vertices of the two parts be denoted by $a^i_1, a^i_2, \ldots, a^i_r$ and $b^i_1, b^i_2, \ldots, b^i_r$ respectively.

We will now do some rewiring of the edges to complete the construction of $\text{GDG}_{k,r}$. Recall that $r - k$ is even and positive. For each vertex c_i of the clique, add one edge from c_i to each of $a^i_1, a^i_2, \ldots, a^i_{\frac{r-k}{2}}$. Similarly, add an edge from c_i to each of $b^i_1, b^i_2, \ldots, b^i_{\frac{r-k}{2}}$. Now remove

the edges $a_1^i b_1^i, a_2^i b_2^i, \ldots, a_{\frac{r-k}{2}}^i b_{\frac{r-k}{2}}^i$. Once this is done for every $i \in [k-1]$, the construction is complete. See Figure 1.

We observe the following property of vertices a_j^i, b_j^i, and c_i of $\textsc{gdg}_{k,r}$.

▶ **Observation 18.** *For every $i \in [k-1]$ and $j \in [r]$, it holds that the degrees of a_j^i and b_j^i in $\textsc{gdg}_{k,r}$ are both exactly r, whereas the degree of c_i is $r-1$.*

We are now ready to prove that Clique in r-regular Graph is many-one reducible to Partial Complement to r-Regular.

▶ **Algorithm 19** (Reduction KrR to PCrR)**.**
Input: An instance (G, k) of KrR.
Output: An instance (G', r) of PCrR such that it is a yes-instance if and only if (G, k) is a yes-instance of KrR.

1. If $k < 7$ or $k \geq r$, solve the instance of KrR by brute force. If it is a yes-instance, return a trivial yes-instance to PCrR, if it is a no-instance, return a trivial no-instance to PCrR.
2. If $r - k$ is odd, modify G by taking two copies of G which are joined by a perfect matching between corresponding vertices. Then r increase by one, whereas k remains the same.
3. Construct the graph G' by taking the disjoint union of G and the gadget $\textsc{gdg}_{k,r}$. Here, r denotes the regularity of G after step 2 is performed. Return (G', r).

Let $n = |V(G)|$. We observe that the number of vertices in the returned instance is at most $2n + (k-1) + 2r \cdot (k-1)$, which is $\mathcal{O}(n^2)$. The running time of the algorithm is $\mathcal{O}(n^7)$ and thus is polynomial.

The correction of the reduction follows from the following two lemmata.

▶ **Lemma 20.** *Let (G, k) be the input of Algorithm 19, and let (G', r) be the returned result. If (G, k) is a yes-instance to* Clique in r-regular Graph*, then (G', r) is a yes-instance of* Partial Complement to r-Regular*.*

Proof. Let $C \subseteq V(G)$ be a clique of size k in G. If the clique is found in step 1, then (G', r) is a trivial yes-instance, so the claim holds. Thus, we can assume that the graph G' was constructed in step 3. If G was altered in step 2, we let C be the clique in one of the two copies that was created. Let $S \subseteq V(G')$ consist of the vertices of C as well as the vertices of the clique K_{k-1} of the gadget $\textsc{gdg}_{k,r}$. We claim that S is a valid solution to (G', r).

We show that $G' \oplus S$ is r-regular. Any vertex not in S will have the same number of neighbors as it had in G'. Since the only vertices that weren't originally of degree r were those in the clique K_{k-1}, all vertices outside S also have degree r in $G' \oplus S$. What remains is to examine the degrees of vertices of C and of K_{k-1}.

Let c_i be a vertex of K_{k-1} in G'. Then c_i lost its $k-2$ neighbors from K_{k-1}, gained k neighbors from C, and kept $r-k$ neighbors in $K_{r,r}^i$. We see that its new neighborhood has size $k + r - k = r$.

Let $u \in C$ be a vertex of the clique from G. Then u lost $k-1$ neighbors from C, gained $k-1$ neighbors from K_{k-1}, and kept $r-(k-1)$ neighbors from $G - C$. In total, u will have $r-(k-1)+(k-1) = r$ neighbors in $G' \oplus S$. Since every vertex of $G' \oplus S$ has degree r, it is r-regular, and thus (G', r) is a yes-instance. ◀

▶ **Lemma 21.** *Let (G, k) be the input of Algorithm 19, and let (G', r) be the returned result. If (G', r) is a yes-instance to* Partial Complement to r-Regular*, then (G, k) is a yes-instance of* Clique in r-regular Graph*.*

Proof of Lemma 21 is omitted due to space constraints. Lemmata 20 and 21 together with Proposition 16 conclude the proof of NP-hardness. Membership in NP is trivial, so NP-completeness holds. ◀

We remark that if r is a constant not given with the input, the problem becomes polynomial time solvable by Theorem 6.

7 Conclusion and open problems

In this paper we initiated the study of Partial Complement to $\mathcal{G}$. Many interesting questions remain open. In particular, what is the complexity of the problem when $\mathcal{G}$ is

- the class of chordal graphs,
- the class of interval graphs,
- the class of graph excluding a path P_5 as an induced subgraph,
- the class graphs with max degree $\leq r$, or
- the class of graphs with min degree $\geq r$

More broadly, it is also interesting to see what happens as we allow more than one partial complementation; how quickly can we recognize the class $\mathcal{G}^{(k)}$ for some class $\mathcal{G}$? It will also be interesting to investigate what happens if we combine partial complementation with other graph modifications, such as the Seidel switch.

References

1. Alexandre Blanché, Konrad K. Dabrowski, Matthew Johnson, Vadim V. Lozin, Daniël Paulusma, and Viktor Zamaraev. Clique-width for graph classes closed under complementation. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 73:1–73:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.73.
2. André Bouchet. Recognizing locally equivalent graphs. *Discrete Mathematics*, 114(1-3):75–86, 1993. doi:10.1016/0012-365X(93)90357-Y.
3. Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.
4. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.
5. Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.
6. Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by seese. *J. Comb. Theory, Ser. B*, 97(1):91–126, 2007. doi:10.1016/j.jctb.2006.04.003.
7. Andrzej Ehrenfeucht, Jurriaan Hage, Tero Harju, and Grzegorz Rozenberg. Complexity issues in switching of graphs. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, November 16-20, 1998, Selected Papers*, volume 1764 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 1998. doi:10.1007/978-3-540-46464-8_5.

8 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003. URL: http://epubs.siam.org/sam-bin/dbq/article/38405.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

11 Vít Jelínek, Eva Jelínková, and Jan Kratochvíl. On the hardness of switching to a small number of edges. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, volume 9797 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2016. doi:10.1007/978-3-319-42634-1_13.

12 Eva Jelínková and Jan Kratochvíl. On switching to H-free graphs. *Journal of Graph Theory*, 75(4):387–405, 2014. doi:10.1002/jgt.21745.

13 Eva Jelínková, Ondrej Suchý, Petr Hlinený, and Jan Kratochvíl. Parameterized problems related to seidel's switching. *Discrete Mathematics & Theoretical Computer Science*, 13(2):19–44, 2011. URL: http://dmtcs.episciences.org/542.

14 Marcin Kaminski, Vadim V. Lozin, and Martin Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009. doi:10.1016/j.dam.2008.08.022.

15 Jan Kratochvíl. Complexity of hypergraph coloring and seidel's switching. In Hans L. Bodlaender, editor, *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*, pages 297–308. Springer, 2003. doi:10.1007/978-3-540-39890-5_26.

16 Jan Kratochvíl, Jaroslav Nešetřil, and Ondřej Zýka. On the computational complexity of Seidel's switching. In *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity (Prachatice, 1990)*, volume 51 of *Ann. Discrete Math.*, pages 161–166. North-Holland, Amsterdam, 1992. doi:10.1016/S0167-5060(08)70622-8.

17 Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005. doi:10.1016/j.jctb.2005.03.003.

18 Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017. doi:10.1016/j.dam.2016.08.006.

19 J. J. Seidel. Graphs and two-graphs. In *Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1974)*, volume X of *Congressus Numerantium*, pages 125–143. Utilitas Math., Winnipeg, Man., 1974.

20 J. J. Seidel. A survey of two-graphs. In *Colloquio Internazionale sulle Teorie Combinatorie (Rome, 1973), Tomo I*, volume 17 of *Atti dei Convegni Lincei*, pages 481–511. Accad. Naz. Lincei, Rome, 1976.

21 J. J. Seidel and D. E. Taylor. Two-graphs, a second survey. In *Algebraic methods in graph theory, Vol. I, II (Szeged, 1978)*, volume 25 of *Colloq. Math. Soc. János Bolyai*, pages 689–711. North-Holland, Amsterdam-New York, 1981.

New Algorithms for Distributed Sliding Windows

Sutanu Gayen[1]
University of Nebraska-Lincoln
Lincoln NE, USA
sutanugayen@gmail.com

N. V. Vinodchandran[2]
University of Nebraska-Lincoln
Lincoln NE, USA
vinod@cse.unl.edu

Abstract

Computing functions over a distributed stream of data is a significant problem with practical applications. The *distributed streaming model* is a natural computational model to deal with such scenarios. The goal in this model is to maintain an approximate value of a function of interest over a data stream distributed across several computational nodes. These computational nodes have a two-way communication channel with a *coordinator node* that maintains an approximation of the function over the entire data stream seen so far. The resources of interest, which need to be minimized, are communication (primary), space, and update time. A practical variant of this model is that of *distributed sliding window (dsw)*, where the computation is limited to the last W items, where W is the window size. Important problems such as sampling and counting have been investigated in this model. However, certain problems including computing frequency moments and metric clustering, that are well studied in other streaming models, have not been considered in the distributed sliding window model.

We give the first algorithms for computing the frequency moments and metric clustering problems in the distributed sliding window model. Our algorithms for these problems are a result of a general transfer theorem we establish that transforms any algorithm in the distributed infinite window model to an algorithm in the distributed sliding window model, for a large class of functions. In particular, we show an efficient adaptation of the smooth histogram technique of Braverman and Ostrovsky, to the distributed streaming model. Our construction allows trade-offs between communication and space. If we optimize for communication, we get algorithms that are as communication efficient as their infinite window counter parts (upto polylogarithmic factors).

2012 ACM Subject Classification Theory of computation → Streaming models, Theory of computation → Sketching and sampling, Theory of computation → Distributed algorithms

Keywords and phrases distributed streaming, distributed functional monitoring, distributed sliding window, frequency moments, k-median clustering, k-center clustering

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.22

1 Introduction

Modern data often arrive fast and are distributed over several nodes. In such situations it is not practical to store and process all the data at a central location. The *distributed streaming* model is a natural architecture for such computing scenario. In this model, a set

[1] Research Funded in part by NSF grant CCF-1422668 and UNL Layman Grant
[2] Research Funded in part by NSF grant CCF-1422668 and UNL Layman Grant

of distributed computational nodes get a stream of data items. Each distributed node has a two-way communication channel to a *coordinator node*. The goal of the coordinator node is to continuously keep track of an approximate value of a function of interest over the union of all streams distributed over the nodes. As in the case of traditional streaming situation, it is assumed that a data item cannot be revisited unless stored on local memory. The primary resource of interest is the total number of bits of *communication* between the coordinator node and distributed nodes. Other important resources are total space used over all nodes and the update time per data item.

The focus of this paper is the *sliding window* variation of this distributed streaming model: *distributed sliding window (dsw)* model. In this model, the goal of the coordinator node is to continuously compute an approximation of a desired function over *only the last W data items* in the union of all data items arriving at each of the distributed nodes. Example of practical situations where such computing scenario arise include geographically distributed e-commerce servers or LAN devices or sensor nodes trying to compute some statistics about the transactions made or the traffic passed or the interesting events happened respectively, for the last million items or during last 24 hours. In general the sliding window variations are more difficult for algorithm design than their infinite window counter parts because of 'implicit deletion': the last element of the current window gets deleted at the arrival of the next element. In the distributed model there are additional challenges since the coordinator node is not directly aware of the arrival of a new item.

1.1 Our Results

We design efficient (communication, space, time) algorithms in the distributed sliding window model for functions including frequency moments and metric clustering. Our algorithms for these problems are results of a general transfer theorem we establish that transforms any algorithm in the distributed (infinite window) streaming model (diw) to an algorithm in the distributed sliding window model, for a large class of functions. Specifically, we adapt the *smooth histogram* technique of Braverman and Ostrovsky [5] that applies to the class of *smooth functions* in the single stream model, to the distributed setting. In particular, we prove the following transfer theorem (please refer to the next section for definitions and notations).

▶ **Theorem.** *Let f be an (α, β)-smooth function f for some $0 < \beta < \alpha < 1$. Let $0 < \epsilon < 1$ be such that $b = \frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta) < 1$. Fix any $0 \le x, y \le 1, x + y = 1$. Suppose there is a diw algorithm $\mathcal{B}$ computes f over stream size at most m, upto approximation ratio $(1 \pm \epsilon)$, using cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then there is another algorithm that computes f over a dsw of size W upto approximation ratio $(1 \pm (\alpha+\epsilon))$ using cost $\langle L \cdot W^x \cdot (1+\log W) c_{\mathcal{B}}(W,\epsilon), (4L \cdot s_{\mathcal{B}}(W,\epsilon) + W^y), L(4+\log W) t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L = ((\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2)$, $f_{\max}$ = the maximum value of f over any window of size W, and $f_{\min}$ = smallest non-zero value of f. We assume storing each data element takes unit space.*

The construction allows trade-offs between communication and space. In particular setting $x = y = 1/2$, we get that for any function f with $f_{\max}/f_{\min} = \text{poly}(W)$, a diw algorithm for f with cost $\langle c, s, t\rangle$ can be transformed to get a dsw algorithm with cost $\langle \tilde{O}(\sqrt{W}c), \tilde{O}(\sqrt{W}+s), \tilde{O}(t)\rangle$, where $\tilde{O}$ hides polylog factors.

We apply this general algorithm for computing frequency moments and metric clustering problems to get new algorithms that are as communication efficient as their infinite window counter parts (upto polylog factors). Although the cost functions associated with clustering problems are not exactly smooth, we still give dsw algorithms for them based on the single stream clustering algorithms given in [4].

1.2 Previous work

Initial research in the distributed streaming model appeared in papers including [2, 10, 15, 18, 19]. These papers designed distributed streaming algorithms for several natural problems including approximately tracking the functions: sum, top-k frequencies, set-expression cardinality, approximate quantiles and thresholded counts. We refer the reader to the above papers for practical motivations behind this model. A formal algorithmic approach towards distributed streaming was first given by Cormode, Muthukrishanan and Yi [11]. They called their model *distributed functional monitoring model* where the task is to continuously monitor whether the function value is $\geq \tau$ or $\leq (1-\epsilon)\tau$ for a threshold τ and an error parameter[3] ϵ. They designed the first algorithm for F_p, the p^{th} frequency moment and also proved lower bound results on the communication cost of monitoring F_p for $p \leq 2$. Efficient randomized algorithms for monitoring count, frequencies and ranks were given in [17]. Woodruff and Zhang [22] designed better algorithms for F_p and provided matching lower bound for the communication cost. Sampling algorithms over distributed streams were given in [12, 21]. Recently Chen and Zhang [7] gave an algorithm for distributed monitoring of entropy. We refer the reader to [9] for a survey by Cormode on this topic.

While the "infinite-window" distributed streaming has received considerable attention, its sliding (or finite) window counterpart has received only limited attention. The first paper to deal with distributed stream processing over a finite sliding window is [12] where the authors present algorithms for sampling that is efficient in communication, space and time. Later, in [6, 14], efficient algorithms for distributed sliding window were designed for the problems of counting the number of bits, quantiles, and heavy hitters. In [20], the authors extend the count-min sketch algorithm to the dsw model. In [16], the present authors give an algorithm for *Euclidean* k-median clustering problem. To the best of our knowledge computations of F_p for a general p and metric clustering are not yet considered in this model (which we consider in this paper). We would like to note that F_p computation and clustering problems have received considerable attention in the *single stream sliding window models* [1, 3, 4, 5, 8].

2 Background and definitions

2.1 The models

The distributed streaming model. In the distributed streaming model there are $(K+1)$ computational nodes: $\{N_1, N_2, \ldots, N_K, C\}$ where N_is are called *distributed* nodes and C is called the *coordinator* node. These nodes have to collectively compute a function f over a global stream of data items: $\{d_1, d_2, \ldots, d_t, \ldots, d_N\}$ which are distributed over N_is in an arbitrary manner. More precisely, at time t, the item d_t will be sent to the node N_j for some $1 \leq j \leq m$. At all times t, the coordinator should maintain an approximation of f over the set of items $\{d_1, d_2, \ldots, d_t\}$ seen so far from the global stream. In order to achieve this, each N_j can communicate with C through a bi-directional channel. An algorithm in this model must work for any ordering of the global input stream. The resources of interest are the total communication, total space usage over all nodes, and time to process each data item. The term *local stream* will be used to refer to the sub-stream seen only at a particular node N_j. In this paper we call this model is the *distributed infinite window* model.

[3] This does not loose generality as the authors observed that any monotonic function f can be computed continuously using $\tilde{O}(\frac{1}{\epsilon})$ copies of a monitoring algorithm for f.

The distributed sliding window model. In the sliding window variation of the distributed infinite window model, there are the global stream and the set of $(K+1)$ nodes as before. But at any time t, the coordinator needs to maintain an approximation of the function f over the set of *most recent* W data items: $\{d_{t-W+1}, \ldots, d_t\}$. This set of items is known as the *active window* and W is known as the *window size*. As in most of the prior literature, we assume each data item comes with a unique (modulo W) time-stamp[4].

2.2 Smooth functions and smooth histograms

The notion of smooth functions was introduced by Braverman and Ostrovsky in [5]. The main property that a smooth function f should satisfy is the following continuity property: Consider f computed on a stream starting at two time points (or indices) i and j ($j > i$). If f computed starting at i and f computed starting at j are within a constant factor of each other at a given point in time, then it will remain within a constant factor in the future.

▶ **Definition 1** ((α, β)-smooth function [5])**.** A function f defined on a set χ of elements is called (α, β)-smooth, for some $0 < \beta < \alpha < 1$ if (1) f is non-decreasing and non-negative, (2) $f(A)$ is at most $poly(|A|)$, (3) $(1-\beta)f(A \cup B) \leq f(B) \implies (1-\alpha)f(A \cup B \cup C) \leq f(B \cup C)$ for any set $A, B, C \subseteq \chi$.

Braverman and Ostrovsky show that for such smooth functions, it suffices to run an online algorithm to compute the function starting at logarithmic number of carefully chosen indices to get constant approximation at any given window. These indices correspond to a constant factor decrease in the value of the function. The corresponding data structure is referred as smooth histogram. This resulted in a construction that translates any single stream infinite window algorithm to a single stream sliding window algorithm for smooth functions with comparable space complexity (up to log factors) as that of the infinite window algorithm.

▶ **Definition 2** (Approximate smooth histogram [5])**.** Let f be an (α, β) smooth function. A smooth histogram for f is a data structure that consists of an increasing set of indices $[I_1, I_2, \ldots, I_L]$ over a sliding window of size W with the following properties.

1. For each $i = 1$ to L, there is an instance of a $(1 \pm \epsilon)$-approximating algorithm A, running for approximating $f(I_i, N)$, the value of f for $\epsilon \leq \beta/4$ over the set $\{d_{I_i}, d_{I_i+1}, \ldots, d_N\}$, where d_j is the data element at location j and d_N is the most recently arrived element.
2. I_1 is expired and I_2 is active or $I_1 = 0$.
3. For $i = 1$ to $t-2$, either 1) $(1-\alpha)f(I_i, N) \leq f(I_{i+1}, N)$ and $(1-\beta/2)f(I_i, N) > f(I_{i+2}, N)$ or 2) $(1-\beta/2)f(I_i, N) > f(I_{i+1}, N)$ and $I_{i+1} = I_i + 1$.

For a smooth function, the third point above ensures that consecutive indices are farthest apart but staying within at least $(1-\alpha)$ factor (or immediate in stream and drops by $> (1-\beta/2)$ factor). Note that $f(I_i, N)$ where I_i is the least index from the histogram that is contained in the current window, approximates the value of the function on the current window.

2.3 Notation

In this paper we use the abbreviations diw and dsw to mean distributed infinite window and distributed sliding window respectively. We denote by $(t_1, t_2]$ the subset of the stream from

[4] This model is also known as time-based dsw model. There is another variation called sequence-based dsw, where no time-stamps are available. This model is harder to design algorithms on.

(t_1+1)-st through t_2-th element. We also use: K to denote the number of distributed nodes, m the length of stream, W the length of window and k the number of medians/centers for clustering. $\langle c, s, t\rangle$ denotes the costs of our diw/dsw algorithm, where c is the communication complexity over any window of size W or over the length of the stream m as appropriate, s is the space complexity and t is the update time (possibly amortized). For a function f, by a c-factor approximation we mean $\frac{f}{c} \leq \tilde{f} \leq c \cdot f$ and by $(1 \pm \epsilon)$ approximation we mean $(1-\epsilon)f \leq \tilde{f} \leq (1+\epsilon)f$.

3 A transfer theorem for smooth functions

In this section we give a general construction that transforms a distributed infinite window (diw) algorithm for smooth function to a distributed sliding window (dsw) algorithm. We assume that the infinite window algorithm $\mathcal{B}$ has a cost-tuple $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$, over a stream of length m and with error parameter ϵ. In particular we prove the following theorem (same as the theorem stated in the introduction).

▶ **Theorem 3.** *Let f be an (α,β)-smooth function f for some $0<\beta<\alpha<1$. Let $0<\epsilon<1$ be such that $b=\frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta)<1$. Fix any $0\leq x,y\leq 1, x+y=1$. Suppose there is a diw algorithm $\mathcal{B}$ computes f over stream size at most m, upto approximation ratio $(1\pm\epsilon)$, using cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then there is another algorithm that computes f over a dsw of size W upto approximation ratio $(1\pm(\alpha+\epsilon))$ using cost $\langle L\cdot W^x\cdot(1+\log W)c_{\mathcal{B}}(W,\epsilon), (4L\cdot s_{\mathcal{B}}(W,\epsilon)+W^y), L(4+\log W)t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L=((\log\frac{f_{\max}}{f_{\min}(1-\epsilon)}/\log\frac{1}{b})+2)$, $f_{\max}=$ the maximum value of f over any window of size W, and $f_{\min}=$ smallest non-zero value of f. We assume storing each data element takes unit space.*

Note that the construction allows trade-offs between communication and space. In particular setting $x=y=1/2$, we get that for any function f with $f_{\max}/f_{\min}=\text{poly}(W)$, a diw algorithm for f with cost $\langle c,s,t\rangle$ can be transformed to get a dsw algorithm with cost $\langle\tilde{O}(\sqrt{W}c), \tilde{O}(\sqrt{W}+s), \tilde{O}(t)\rangle$, where $\tilde{O}$ hides polylog factors.

We first give the algorithm and its proof for the case when $x=0$ and $y=1$, and then point out how to modify this algorithm to get the general algorithm. This algorithm has communication and time costs almost same as (upto polylog(W)) that of the diw algorithm, but uses $\Theta(W)$ space.

▶ **Theorem 4.** *Let f be an (α,β) smooth function for some $0<\beta<\alpha<1$. Let $0<\epsilon<1$ be any number such that $b=\frac{(1+\epsilon)^2}{(1-\epsilon)^2}(1-\beta)<1$. Further assume f has a $(1\pm\epsilon)$-approximate diw algorithm $\mathcal{B}$ over stream size at most m, with cost $\langle c_{\mathcal{B}}(m,\epsilon), s_{\mathcal{B}}(m,\epsilon), t_{\mathcal{B}}(m,\epsilon)\rangle$. Then, there is a dsw algorithm for computing f upto approximation ratio $(1\pm(\alpha+\epsilon))$ with cost $\langle L(\log W+1)c_{\mathcal{B}}(W,\epsilon), L\cdot s_{\mathcal{B}}(W,\epsilon)+W, L(\log W+1)t_{\mathcal{B}}(W,\epsilon)\rangle$, where $L\leq(\log\frac{f_{\max}}{f_{\min}(1-\epsilon)}/\log\frac{1}{b})+2$, $f_{\max}=$ the maximum value of f over any window of size W, and $f_{\min}=$ smallest non-zero value of f. We assume storing each data element takes unit space.*

High level idea of the algorithm: Our general approach is to adapt the smooth histogram technique for sliding windows due to Braverman and Ostrovsky [5] to the distributed setting. Braverman and Ostrovsky showed that for smooth functions, if streaming algorithms are maintained from a small set of carefully chosen indices of the active window, one of these algorithms would approximate the value of the function over all active windows in near future. These indices correspond to a drop of the value of the function by some constant factor. Their algorithm has three main steps: when a new data item d arrives (1) start a new instance of the streaming algorithm $\mathcal{A}$ from this new item (2) update all running instances of

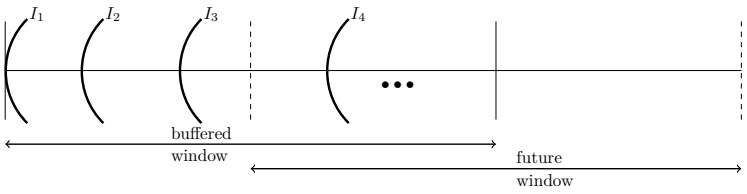


Figure 1 Online algorithms run from each index I_j (position indicated by '('). The one from I_3 can be used to approximate f over the future window. Thus, indices from the buffered window serves for next W elements.

$\mathcal{A}$ with d and (3) remove redundant indices and the corresponding instances of $\mathcal{A}$. There are technical challenges to translate the smooth histogram technique to the distributed setting. The main obstacle is the following: In the single stream case, for each newly arrived element, an instance of $\mathcal{A}$ is started from its index. Most of them are removed at some later point in the future so that at all times only a logarithmic number of indices are kept. In distributed setting, if one has to start instances of $\mathcal{A}$ each time a new element arrives, it will cost $\Omega(W)$ bits of communication (from the distributed node where the item arrives to the coordinator).

In order to reduce the communication cost we use the following approach. Instead of continuously building the histogram, we observe that it is enough to create it once per W items. Once built, this histogram will continue to work till the arrival of next W^{th} item due to smoothness of the function. In other words, steps (1) and (3) in the previous discussion could be dropped except once per W items. This keeps the asymptotic time and communication cost small. We split the entire stream into static windows of size W: $(0, W], (W, 2W], \ldots, (aW, aW + W], \ldots$ and we build the smooth histogram only when the current window coincides with one of the static windows. This is done by buffering the entire expiring window using $O(W)$ space (see Figure 1 for illustration). The indices, which correspond to a drop of some constant factor for f, are obtained by performing binary searches on the buffered static window. This will introduce further communication cost and time for about $O(\log^2 W)$ many instances of the online algorithm.

Proof. (*Of Theorem 4*). A high-level pseudocode of the algorithms is given in Algorithm 1. We split the entire stream into static windows of size W: $(0, W], (W, 2W], \ldots, (aW, aW + W], \ldots$ and maintain a smooth histogram over exactly one of them. If the current time t satisfies $aW - W < t < aW$, the smooth histogram from the static window $(aW - 2W, aW - W]$ can be used to approximate $f([t - W + 1, t])$. We recall, the smooth histogram maintains online algorithms from a set of appropriately chosen indices guarantees that any two consecutive indices are either consecutive or are $(1 - \alpha)$-factor close in f. If $I_1 \leq (t - W + 1) < I_2$ are the two unique consecutive pair of indices enclosing $(t - W + 1)$, the value of the online algorithm from the index I_1 can be used to approximate $f([t - W + 1, t])$ upto $(1 \pm (\alpha + \epsilon))$-factor. We buffer the next static window $(aW, aW + W]$ locally (at the distributed nodes as they arrive) and build the smooth histogram from it at time $(aW + W)$. We describe how to build this in detail in the following claim.

▶ **Claim 5.** *Suppose we have stored a static window $[\lambda, \rho] = (aW, aW + W]$ locally. A smooth histogram $\mathcal{H}$ for $[\lambda, \rho]$ can be built using cost $\langle L \log W \cdot c_{\mathcal{B}}(W, \epsilon), (s_{\mathcal{B}}(W, \epsilon) + W), L \log W \cdot t_{\mathcal{B}}(W, \epsilon)\rangle$, where the number of indices in $\mathcal{H}$ is $L \leq (\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2$.*

Algorithm 1: High level dsw algorithm for smooth functions

Input: A stream of data: $\langle d_1, d_2, \ldots, d_N \rangle$
Output: Approximate value of $f((N-W, N])$

```
while Not the end of stream do
    Let d_N be the newly arrived item;
    if N = aW + W then
        Delete the current smooth histogram H((aW − W, aW]);
        H((aW, aW + W]) ← Build a new smooth histogram for (aW, aW + W];
        Output the value of online algorithm over (aW, aW + W];
    else
        if aW < N < aW + W then
            Update the online algorithm from each index of H((aW − W, aW]) with
             d_N;
            Buffer d_N at the node where it arrived;
            I_j, I_{j+1} be the immediate indices such that I_j ≤ N − W + 1 < I_{j+1};
            Output the value of the online algorithm started from index I_j;
        end
    end
end
```

Proof. We find the indices from the buffered window $[\lambda, \rho]$ by binary search and by running the online algorithm $\mathcal{B}$ on the buffered set of items[5]. The first index is always λ. The last index is always the index ρ. We denote by $\tilde{f}$, a $(1 \pm \epsilon)$ factor approximation for f, found using $\mathcal{B}$. Then, the second index is created at a time-stamp t, such that,

$$\tilde{f}([t, \rho]) \geq (1-\beta)\frac{(1+\epsilon)}{(1-\epsilon)}\tilde{f}([\lambda, \rho]). \tag{1}$$

This ensures, $f([t, \rho]) \geq (1-\beta)f([\lambda, \rho])$, as desired for smooth histogram. In fact, we try to find such a t as far as possible in the window to minimize the number of indices. We tag a time-stamp 'yes' if it satisfies Equation (1) and 'no' otherwise. We set two variables $l = \lambda$ and $r = \rho$ and maintain the invariant that l has 'yes' tag and r has 'no' tag. We next check whether $mid = (l+r)/2$ has 'yes' tag or 'no' tag and update l or r appropriately to maintain the invariant. Then, in $\log W$ steps, we will be able to get hold of a t, such that, t has 'yes' tag but $(t+1)$ has 'no' tag. This is our next index. We find subsequent indices in similar manner.

Notice that, at the $(t+1)$-st item, the value of $\tilde{f}$ drops at least by factor $(1-\beta)\frac{(1+\epsilon)}{(1-\epsilon)}$ (Recall, if the indices are consecutive, drop could be larger). This implies, f drops at least by factor $b = (1-\beta)\frac{(1+\epsilon)^2}{(1-\epsilon)^2} < 1$. Suppose there are L indices in total. Then, After crossing the $(L-1)$-st index, the value of $\tilde{f}$ is at most $f_{\max} b^{(L-2)}$, where $f_{\max}$ is the maximum

[5] There are some details for running $\mathcal{B}$ on the buffered items. For a general function, assume, all the nodes have some global knowledge of time. Then, a fixed time interval of sufficient length can be allotted for processing each data item. Thus, for example, it may be agreed upon that the k^{th} data item d in the current window will be processed during time interval $(t, t + \Delta \cdot k)$ where Δ is at least as large as the update time of the algorithm and t is the time of arrival of the oldest element of the current window. During this interval, whichever node has received d, will process it. For a permutation-invariant function such as F_p and clustering, the nodes can take turns and run the online algorithm over the desired sub-window of the local stream, to compute the function over any sub-window of the global stream.

value of f over any window. Moreover, assuming the least non-zero value of f is $f_{\min}$, $f_{\max}b^{(L-2)} \geq f_{\min}(1-\epsilon)$. Hence $L \leq (\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2$. We denote by I this set of indices. This concludes the updating of the smooth histogram. While finding the indices, at most $L \log W$ instance of $\mathcal{B}$ are run for at most W units of time. This can be achieved using $L \log W \cdot c_{\mathcal{B}}(W, \epsilon)$ total communication and $LW \log W \cdot t_{\mathcal{B}}(W, \epsilon)$ total time (i.e. amortized update time $L \log W \cdot t_{\mathcal{B}}(W, \epsilon)$ per item). During the binary search, we need space for running at most a single instance of $\mathcal{B}$ at any point in time, and the space is reused. We also need space for buffering the current window. So, the space complexity for this part is $(s_{\mathcal{B}}(W, \epsilon) + W)$. ◀

Afterwards, we maintain $\mathcal{B}$ from each of the indices and update them with newly arrived items. For any time till the arrival of next $(W-1)$ items, let $f_1 > f_2$ be the value of f at the two enclosing indices of the current window. Let $f_{current}$ be the value of f over the current window. From the properties of smooth histogram, either $f_{current} = f_1$, or $(1-\alpha)f_1 < f_2 \leq f_{current} \leq f_1$. Moreover, the value of online algorithm at the former index holds a value $\tilde{f}_1$, such that, $(1-\epsilon)f_1 \leq \tilde{f}_1 \leq (1+\epsilon)f_1$. Hence, $(1-\epsilon)f_{current} \leq \tilde{f}_1 \leq \frac{(1+\epsilon)}{(1-\alpha)} f_{current}$. This is close to $(1 \pm (\alpha + \epsilon))$-approximation for small α and ϵ. We also need to continue running $\mathcal{B}$ from each index for W units of time. This costs at most $L \cdot c_{\mathcal{B}}(W, \epsilon)$ communication, $L \cdot t_{\mathcal{B}}(W, \epsilon)$ update time per item and $L \cdot s_{\mathcal{B}}(W, \epsilon)$ space in total. ◀

Proof of Theorem 3. The proof follows closely from that of Theorem 4. In this case, we break the current window of size W into W^x blocks, each of size W^y, such that $W = W^x \cdot W^y$, for some $0 < x, y < 1, x+y = 1$. We rebuild the smooth histogram per arrival of W^x elements in the combined stream. Then, the nodes need to store at most W^x items. As before, total number of indices within each block, $L_1 \leq L = ((\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2)$. As in the algorithm of Theorem 4, finding these indices is done by binary search, using at most $L_1 \log W$ calls to $\mathcal{B}$. In total, this can be done with $L_1 \log W c_{\mathcal{B}}(W, \epsilon)$ total communication per block (i.e $L_1 \cdot W^x \cdot \log W c_{\mathcal{B}}(W, \epsilon)$ in total), $(s_{\mathcal{B}}(W, \epsilon) + W^y)$ space (since space is reused during binary-search) and $L_1 \log W W^y t_{\mathcal{B}}(W, \epsilon)$ total time per block (i.e. amortized $L_1 \log W t_{\mathcal{B}}(W, \epsilon)$ time per item).

Subsequently, we need to maintain $L_1 \cdot W^x$ online algorithms from each of the indices within the current window of size W. But we can do better by removing unnecessary ones while introducing indices from a new block. We arrange the combined indices in decreasing order of arrival. Then, for each index i, we look for the subsequent index j where the current value of the online algorithm drops by factor b for the first time. We remove all indices strictly between i and $(j-1)$ if there are any. We repeat this removal procedure until no more indices can be removed in such a manner. Then, starting from each index, at the next to next index, value of online algorithm drops at least by factor b. After merging, there are $L_2 \leq (2(\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b}) + 2) \leq 2L$ such indices at any point in time. Moreover, by previous discussion, every next index is either the subsequent item, or within a factor b from the previous index. This entire removal takes time linear in the set of the indices to be merged, i.e. at most $10(\log \frac{f_{\max}}{f_{\min}(1-\epsilon)} / \log \frac{1}{b})$ time per block. So, we ignore this while computing the update time per item. Note that, during this removal, no further communication or space is required. This concludes the indices removal procedure. Then, online algorithms from each of the indices run for $\geq W^y$ and $\leq W$ time. So the total communication cost is at most $L_1 \cdot W^x c_{\mathcal{B}}(W, \epsilon)$. The space complexity for running the online algorithms is at most $2L_2 \cdot s_{\mathcal{B}}(W, \epsilon)$. Using the indices removal procedure, we improve the update time to $2L_2 \cdot t_{\mathcal{B}}(W, \epsilon)$ per arrival, for updating each of the current instances of $\mathcal{B}$. ◀

3.1 Better and simpler algorithm for symmetric smooth functions

For symmetric smooth functions we get simpler algorithm with slightly better cost. In particular, the cost of the algorithm will be $\langle (L+1)\cdot c_{\mathcal{B}}(W,\epsilon)\cdot W^x, 4L\cdot s_{\mathcal{B}}(W,\epsilon)+W^y, 4(L+1)t_{\mathcal{B}}(W,\epsilon)\rangle$.

We call a function symmetric if its value is invariant to the permutation of its arguments. For symmetric functions, we create the indices of the smooth histogram by making a single backward pass (i.e. from the most recent to the least recent item in the window) of the distributed online algorithm on the buffered window $[aW+1, aW+W]$. Let $\tilde{f}(A)$ be the value of this algorithm, which is within $(1\pm\epsilon)$-factor of $f(A)$. We create the last index of the smooth histogram at $(aW+W)$. Recursively assume, the previous index we created was at t_1. During the backward pass, suppose at the time-stamp $(t_2-1)\leq t_1$, the value $\tilde{f}([t_2-1, aW+W])$ is at least $\frac{1}{b}\cdot\tilde{f}([t_1, aW+W]) = \frac{1}{(1-\beta)}\frac{(1-\epsilon)}{(1+\epsilon)}\cdot\tilde{f}([t_1, aW+W])$ for the first time. If this happens at $t_2-1=t_1-1$, we create an index at (t_1-1). Otherwise, we create at t_2, which satisfies $\tilde{f}([t_2, aW+W]) < \frac{1}{b}\tilde{f}([t_1, aW+W])$. This implies, $f([t_1, aW+W])\geq(1-\beta)f([t_2, aW+W])$, ensuring the smoothness condition between the consecutive indices t_1 and t_2. We find all the L indices in similar manner, where $L=((\log\frac{f_{\max}}{f_{\min}(1-\epsilon)}/\log\frac{1}{b})+2)$, $f_{\max}$ = the maximum value of f over any window of size W, and $f_{\min}$ = smallest non-zero value of f. This improves the cost of Theorem 4 to $\langle (L+1)c_{\mathcal{B}}(W,\epsilon), (L+1)\cdot s_{\mathcal{B}}(W,\epsilon)+W, (L+1)t_{\mathcal{B}}(W,\epsilon)\rangle$. We can shave off a $\log W$ factor from the costs of Theorem 3 using a similar simpler algorithm. Note that, we crucially use the symmetric nature of the function in the use of the backward online algorithm. The F_p and clustering costs are symmetric functions, whereas, the function 'length of longest increasing subsequence' is asymmetric smooth [5].

4 Applications: Computing F_p and Clustering

In this section we apply the transfer theorem to get new dsw algorithms for approximating F_p and metric clustering problems.

4.1 Computing F_p

▶ **Definition 6** (p^{th} Frequency moment). Given a set of items $\{1,2,\ldots,n\}$ such that their frequencies are $\{f_1,f_2,\ldots,f_n\}$ respectively, their p^{th} frequency moment is defined as $F_p=\sum_{i=1}^{n} f_i^p$.

We first recall a result that shows F_p is smooth.

▶ **Theorem 7** (Lemma 5 of [5]). *Fix any $0<\epsilon<1$. For $p<1$, F_p is (ϵ,ϵ)-smooth function. For $p\geq 1$, F_p is $(\epsilon,\frac{\epsilon^p}{p^p})$-smooth function.*

The first distributed functional monitoring algorithm for F_p was given in [11]. In the monitoring model, in a distributed stream of items, one has to decide whether $F_2\geq\tau$ or $F_2\leq(1-\epsilon)\tau$ at all times, for some ϵ and a threshold τ specified. The same paper ([11]) also observed that any monotonic function f can be computed continuously using $\tilde{O}(\frac{1}{\epsilon})$ copies of a monitoring algorithm for f. Later on Woodruff and Zhang provided the following online algorithm for F_p, for any $p\geq 1$. They also showed this algorithm has optimal dependence on K.

▶ **Theorem 8** (Follows from Theorem 8 of [22]). *For any $0<\epsilon<1$, there is an algorithm that continuously computes F_p for any constant $p\geq 1$, over universe $[n]$ over a distributed stream of length at most W upto approximation $(1\pm\epsilon)$ with high probability using cost $\langle\tilde{O}(\frac{K^{p-1}}{\epsilon^{\Theta(p)}}),\tilde{O}(\frac{nK}{\epsilon}),\tilde{O}(\frac{1}{\epsilon^2})\rangle$.*

From Theorem 3, we get the following algorithm for computing F_p, which is $(\epsilon, \epsilon^p/p^p)$-smooth.

▶ **Corollary 9.** *Fix any $0 \leq x, y \leq 1, x + y = 1$. For any constant p, there is an algorithm that continuously computes F_p over a time based dsw of width W upto approximation ratio $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}(W^x \frac{K^{p-1}}{\epsilon^{\Theta(p^2)}}), \tilde{O}(W^y + \frac{nK}{\epsilon^{\Theta(p)}}), \tilde{O}(\frac{1}{\epsilon^{\Theta(p)}}) \rangle$.*

In particular for F_2, we chose to work with the following result of [11] since its communication cost has much smaller dependence on ϵ.

▶ **Theorem 10** (Follows from Theorem 6.1 of [11])**.** *For any $0 < \epsilon < 1$, there is an algorithm that continuously computes F_2 over universe n over a distributed stream of length at most W upto approximation $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}((\frac{K^2}{\epsilon^2} + \frac{K^{1.5}}{\epsilon^4})), \tilde{O}(\frac{K}{\epsilon^3}), \tilde{O}(\frac{1}{\epsilon^3}) \rangle$.*

From Theorem 3, we get the following algorithm for computing F_2, which is $(\epsilon, \epsilon^2/4)$-smooth.

▶ **Corollary 11.** *Fix any $0 \leq x, y \leq 1, x + y = 1$. There is an algorithm that continuously computes f over a time based dsw of width W upto approximation ratio $(1 \pm \epsilon)$ with high probability using cost $\langle \tilde{O}(W^x(\frac{K^2}{\epsilon^4} + \frac{K^{1.5}}{\epsilon^8})), \tilde{O}(W^y + \frac{K}{\epsilon^6}), \tilde{O}(\frac{1}{\epsilon^6}) \rangle$*

4.2 Metric clustering

In this section we apply the generic algorithm from Section 3 for the functions: k-median and k-center clustering.

▶ **Definition 12** (Metric k-median clustering problem)**.** Given a set of points P from a metric space χ, output $C^* = \arg\min_{C \subseteq \chi, |C| \leq k} \sum_{p \in P} \min_{c \in C} d(p, c)$ and $\mathrm{OPT}_k = \sum_{p \in P} \min_{c \in C^*} d(p, c)$ where d is the distance function of χ.

▶ **Definition 13** (Metric k-center clustering problem)**.** Given a set of points P from a metric space χ, output $C^* = \arg\min_{C \subseteq \chi, |C| \leq k} \max_{p \in P} \min_{c \in C} d(p, c)$ and $\mathrm{OPT}_k = \max_{p \in P} \min_{c \in C^*} d(p, c)$ where d is the distance function of χ.

In this section, approximation ratio of $r > 1$ will mean the clustering cost of the algorithm is in the range $[\mathrm{OPT}, r.\mathrm{OPT}]$. We assume each point takes $O(1)$ space. These two clustering problems have approximation ratio $\Theta(1)$. So a straightforward combination of the local clusterings results in an overall approximation ratio of $O(m)$. The cost functions OPT_k are neither smooth [4].

4.2.1 Metric k-median clustering

We use the following diw algorithm for k-median clustering.

▶ **Theorem 14** (Theorem 2 of [16] restated)**.** *There is a distributed online algorithm for $O(1)$-approximate metric k-median with success probability $(1 - \frac{1}{\mathrm{poly}(W)})$, and with cost $\langle O(kK \log^3 W), O(kK \log W)), O(k \log W) \rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

We cannot directly apply Theorem 3 since the k-median clustering cost is not smooth. We use an additional property of k-median cost observed in [4] and use ideas from Theorem 4 to get the following theorem, whose proof is in Section 5.

▶ **Theorem 15.** *There is a dsw algorithm for $O(1)$-approximate metric k-median with success probability $(1 - \frac{1}{\mathrm{poly}(W)})$ per W items, and with cost $\langle O(k^2 K \log^5 W), O(k^2 K \log^3 W + W), O(k^2 \log^3 W) \rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

4.2.2 Metric k-center clustering

We use the following diw algorithm for k-center clustering.

▶ **Theorem 16** (Theorem 6 of [13] restated). *For any $\epsilon > 0$, there is a deterministic distributed online algorithm for $(2+\epsilon)$-approximate metric k-center with cost $\langle O(\frac{kK}{\epsilon}\log W), O(\frac{kK}{\epsilon}\log \mathrm{OPT}), O(k)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

The k-center clustering cost is not smooth. We prove an additional property of k-median cost and use ideas from Theorem 4 to get the following theorem, whose proof is in Section 5.

▶ **Theorem 17.** *There is a deterministic dsw algorithm for $O(1)$-approximate metric k-center with cost $\langle O(k^2K\log^4 W), O(k^2K\log^2 W + W), O(k^2\log^2 W)\rangle$ assuming $\mathrm{OPT}_k = \mathrm{poly}(W)$.*

5 Proofs of clustering results

It was shown in [4], the k-median cost behaves like a smooth function if the following additional property is satisfied. For convenience, we abuse the notation of Definition 1 for a smooth function in this subsection, by replacing $(1-\alpha)$ by $\frac{1}{\alpha}$ and $(1-\beta)$ by $\frac{1}{\beta}$ for appropriate $\alpha > \beta > 1$. We denote by $Cost(P, O)$ the k-median clustering cost for a set of points P, when O is the set of k medians. For convenience, we drop the k in OPT_k when there is no ambiguity.

▶ **Lemma** (Lemma 3.1 of [4] restated). *For any distinct sets of points $A, B, C \subseteq \chi$ from some metric space χ, $\mathrm{OPT}(A\cup B) \le \gamma\mathrm{OPT}(B) \implies \mathrm{OPT}(A\cup B\cup C) \le (2+r\gamma)\mathrm{OPT}(B\cup C)$ for any $r, \gamma \ge 1$, provided the following property holds for the sets A, B: There exists a k-median clustering $t : (A\cup B) \to F$ upto approximation ratio r such that $|t^{-1}(f)\cap A| \le |t^{-1}(f)\cap B|$ for each median $f \in F$.*

▶ **Theorem.** *There is a dsw algorithm for $O(1)$-approximate metric k-median with success probability $(1-\frac{1}{\mathrm{poly}(W)})$ per W items, and with cost $\langle O(k^2m\log^5 W), O(k^2m\log^3 W + W), O(k^2\log^3 W)\rangle$ assuming $\mathrm{OPT} = \mathrm{poly}(W)$.*

Proof. We split the stream into static windows of the form: $[aW+1, aW+W]$ and store this window locally. At time $(aW+W)$, we need to rebuild a smooth histogram. For this, we use the slightly better and simpler algorithm from the remark in Section 3.1. We run $\mathcal{A}$ from Theorem 14 backwards (i.e. from item $(aW+W)$ to item $(aW+1)$). Let the approximation factor of $\mathcal{A}$ be λ. The last index of the smooth histogram is at time-stamp $(aW+W)$. Suppose, the last time we created an index at time-stamp t_l and the value of $\mathcal{A}$ at t_l was $v_l \in [\mathrm{OPT}, \lambda\mathrm{OPT}]$. Fix any $\gamma > \lambda$. Let $(t_{l-1}-1)$ be the time when value of $\mathcal{A}$ is at least $\frac{\gamma}{\lambda}v_l$ for the first time. If $t_{l-1}-1 = t_l - 1$, we create the next index at (t_l-1). Otherwise at t_{l-1}, $\mathcal{A}([t_{l-1}, aW+W]) < \frac{\gamma}{\lambda}v_l$. This implies, $\mathrm{OPT}([t_{l-1}, aW+W]) \le \gamma\cdot\mathrm{OPT}([t_l, aW+W])$. This t_{l-1} is our next index. In total, at most L such indices will be created, where $L = ((\log\frac{\lambda f_{\max}}{f_{\min}})/\log\frac{\gamma}{\lambda}) + 2) = O(\log W)$, $f_{\max}$ = the maximum value of k-median cost over any window of size W, and $f_{\min}$ = smallest non-zero value of k-median cost. We refer to these set of indices as 'outer' indices. For each outer index, we also record the set of k medians produces by $\mathcal{A}$. Let the indices be $I = \{I_1 < I_2 < \cdots < I_L\}$ and the corresponding sets of k medians be $C = \{C_1, C_2, \ldots, C_L\}$, where each $C_i = \{c_{i1}, c_{i2}, \ldots, c_{ik}\}$. We communicate I and C to each node. We claim that for any $I_i \le t < I_{i+1}$, C_i is a $\gamma\lambda$-approximate set of k-medians for $[t, aW+W]$. This is because, $Cost([t, aW+W], C_i) \le Cost([I_i, aW+W], C_i) \le \lambda\cdot OPT([I_i, aW+W]) \le \gamma\lambda\cdot OPT([I_{i+1}, aW+W]) \le \gamma\lambda\cdot OPT([t, aW+W])$ (Using monotonicity of OPT and smoothness).

We also need to ensure the additional property from Lemma 5. We ensure this by keeping a set of 'inner' indices between each pair of outer indices. We describe below how to find the inner indices between I_1 and I_2. Other inner indices can be found accordingly. For any $i = 1$ to L, and for any set $S = [t, aW + W] \subseteq [I_i, aW + W]$, let n_{ij}^S denote the number of points from S which map to the median c_{ij}, in the clustering $[I_i, aW + W] \to C_i$. Let $\tilde{n}_{ij}^S$ denote a $(1 \pm \frac{1}{10})$ approximation of n_{ij}^S. We first assume the coordinator can compute $\tilde{n}_{ij}^S$ for any $S = [t, aW + W] \subseteq [I_i, aW + W]$. We defer the description of how to compute this in the following paragraph. Let J_1 be the first (earliest in window) inner index between I_1 and I_2. We will create the index J_1 at the farthest time-stamp $I_1 < t \leq I_2$, such that

$$\forall j = 1 : k, \tilde{n}_{1j}^{[I_1, aW+W]} \leq (18/11) \cdot \tilde{n}_{1j}^{[t, aW+W]} \tag{2}$$

The later condition ensures, for each median $c_{1j} \in C_1$, $n_{1j}^{[I_1, aW+W]} \leq 2 \cdot n_{1j}^{[t, aW+W]}$, equivalently $n_{1j}^{[I_1, t-1]} \leq n_{1j}^{[t, aW+W]}$, as demanded in the additional property from Lemma 5. Such a farthest t satisfying Equation 2 is found by using binary search. Notice that $t = l := I_1$ is always satisfied. We first guess $t = r := I_2$. If this t satisfies Equation 2, we already have an index at I_2 and we don't need to keep any inner index. If not, we next guess $t = \lceil \frac{l+r}{2} \rceil$ and if this t satisfies, we change $l = \lceil \frac{l+r}{2} \rceil$, otherwise, we change $r = \lceil \frac{l+r}{2} \rceil$. In this way, we preserve the invariant that l satisfies Equation 2 but r does not. In $O(\log W)$ steps, we will get a t^*, such that t^* satisfies but $(t^* + 1)$ does not. We set $J_1 = t^*$. We find the next inner index J_2 similarly using the same clustering C_1 and at the farthest time-stamp t, such that, $\forall j = 1 : k, \tilde{n}_{1j}^{[J_1, aW+W]} \leq \frac{18}{11} \tilde{n}_{1j}^{[t, aW+W]}$ holds. Note that, the set C_1 is a $\gamma\lambda$ approximate median for any $I_1 \leq t < I_2$, from previous discussion. So, the additional property from Lemma 5 holds at indices J_1 and J_2, with respect to the clustering C_1 and $r = \gamma\lambda$. Let C_1' and C_2' be the λ-approximate clusterings for J_1 and J_2 respectively. Hence from Lemma 5 at any later time $(t' + W - 1)$, such that $J_1 \leq t' < J_2$, $Cost([t, t+W-1], C_1') \leq Cost([J_1, t+W-1], C_1') \leq \lambda \cdot OPT([J_1, t+W-1]) \leq \lambda(2 + \gamma^2\lambda) \cdot OPT([I_{i+1}, t + W - 1]) \leq \lambda(2 + \gamma^2\lambda) \cdot OPT([t, t + W - 1])$ (Using monotonicity of OPT and smoothness). So, the final approximation is $(2 + \gamma^2\lambda)$. We find subsequent inner indices in similar manner. Note that, after crossing each inner index, $\tilde{n}_{ij}$ for some j reduces by a factor $\frac{18}{11}$. Since there are at most k medians and W items, the total number of inner indices between I_1 and I_2 is at most $O(k \log W)$. Also note that, for checking Equation 2, the coordinator needs $\tilde{n}_{ij}^{[t, aW+W]}$ values, for various values of t, and j, which we obtain as follows.

Each node makes a backward pass over its local data. During this pass, for each i, it maps each point $p \in [I_i, aW + W]$ to c_{ij^*}, where $j^* = \arg\min_j d(p, c_{ij})$, i.e. c_{ij^*} is the closest of the medians from C_i. It also keeps a counter n_{ij} for each c_{ij}, which increments for each new point mapping to c_{ij}. We then record the time-points where n_{ij} increases by $(1 + \frac{1}{20})$-factor, i.e crosses $\{\frac{21}{20}, \frac{21}{20}^2, \ldots, W\}$ for the first time. We call this set of time-points as H_{ij}. Each node z sends such H_{ij}^z, for each i, j to the coordinator. Note that, $n_{ij}^{[t, aW+W]} = \sum_z n_{ij}^{[t, aW+W]_z}$, where $[t, aW + W]_z$ denotes items from $[t, aW + W]$ that appear at node z. Using H_{ij}^z, coordinator can approximate $n_{ij}^{[t, aW+W]_z}$ upto $(1 \pm \frac{1}{10})$ factor, for any z. Taking sum over all z, it can approximate $n_{ij}^{[t, aW+W]}$ for any t, upto $(1 \pm \frac{1}{10})$ factor, as required.

The total number of indices are $O(kL \log W)$. The backward online algorithm costs $\langle O(km \log^3 W), O(km \log W)), O(k \log W) \rangle$. Communicating the H_{ij}^z values cost $O(kmL \log^3 W)$ in total. The set of inner indices require $O(kL \log^2 W)$ computations for n_{ij}^S per W items by the coordinator, and we ignore the costs for these. Later, online algorithms are run for each index for at most W arrivals. We also have to include $O(W)$ space complexity for

storing the static windows. The final cost is $\langle O(k^2 mL \log^4 W), O(k^2 mL \log^2 W + W), O(k^2 L \log^2 W)\rangle$. Since we run at most $O(kL \log W)$ online algorithms, success probability per W items is still $(1 - \frac{1}{\text{poly}(W)})$. ◀

Next we present a Lemma analogous to Lemma 5 for smoothness of k-center clustering. The additional property is more relaxed than that of k-median. To the best of our knowledge, this result was not known before. We denote by $Cost(P, O)$ the k-center clustering cost for a set of points P, when O is the set of k centers. For convenience, we drop the k in OPT_k when there is no ambiguity.

▶ **Lemma.** *For any distinct sets of points $A, B, C \subseteq \chi$ from some metric space χ, $\text{OPT}(A \cup B) \leq \gamma \text{OPT}(B) \implies \text{OPT}(A \cup B \cup C) \leq (1+2r\gamma)\text{OPT}(B \cup C)$ for any $r, \gamma \geq 1$, provided the following property holds for the sets A, B: There is a k-center clustering $t : (A \cup B) \to F$, upto approximation ratio r, such that, for each center $f \in F, |t^{-1}(f) \cap A| > 0 \implies |t^{-1}(f) \cap B| > 0$.*

Proof. Let O be the optimal set of centers for $B \cup C$. We will map each element $a \in A$ to some point in O. Let O' be the r-approximate set of centers for $A \cup B$, which satisfies the above property. Let $o' \in O'$ be the center to which $a \in A$ maps to. By assumption some $b \in B$ also gets mapped to o'. Finally, let $o \in O$ be the center to which b maps to. We will map a to o. Then by definition, $\max(d(a, o'), d(b, o')) \leq r \cdot \text{OPT}(A \cup B)$ and $d(b, o) \leq \text{OPT}(B \cup C)$. Then, by triangle inequality, $d(a, o) \leq (d(a, o') + d(b, o') + d(b, o)) \leq (2r \cdot \text{OPT}(A \cup B) + \text{OPT}(B \cup C))$

$$
\begin{aligned}
OPT(A \cup B \cup C) &\leq Cost(A \cup B \cup C, O) \\
&\leq \max\{\text{OPT}(B \cup C), Cost(A, O)\} \\
&\leq \max\{\text{OPT}(B \cup C), (2r \cdot \text{OPT}(A \cup B) + \text{OPT}(B \cup C))\} \\
&\leq (2r \cdot \text{OPT}(A \cup B) + \text{OPT}(B \cup C)) \\
&\leq (2r\gamma \cdot \text{OPT}(B) + \text{OPT}(B \cup C)) && \text{(Given)} \\
&\leq (1 + 2r\gamma)\text{OPT}(B \cup C) && \text{(Using monotonicity of OPT)}
\end{aligned}
$$

◀

Our dsw algorithm for k-center clustering closely follows that for k-median clustering given earlier in this section. We create a set of 'outer' indices corresponding to a constant factor drop of the cost of the diw algorithm. We also introduce a set of 'inner' indices between each pair of outer indices to satisfy the additional property of Lemma 5. These inner indices are created at a point t, such that there exists a center to which no item from part $(t, aW + W]$ maps. Since, there are at most k centers, at most k inner indices are possible between each pair of outer indices. Hence the total number of indices is $O(k \log W)$. We skip more details of the proof since it closely follows that of Theorem 15.

References

1. Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the Twenty-Second Symposium on Principles of Database Systems PODS*, pages 234–243, 2003.
2. Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 28–39, 2003.
3. Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering on sliding windows in polylogarithmic space. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 350–364, 2015.

4 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1374–1390, 2016.

5 Vladimir Braverman and Rafail Ostrovsky. Effective computations on sliding windows. *SIAM J. Comput.*, 39(6):2113–2131, 2010.

6 Ho-Leung Chan, Tak Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. Continuous monitoring of distributed data streams over a time-based sliding window. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS*, pages 179–190, 2010.

7 Jiecao Chen and Qin Zhang. Improved algorithms for distributed entropy monitoring. *Algorithmica*, 78(3):1041–1066, Jul 2017.

8 Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, pages 19:1–19:12, 2016.

9 Graham Cormode. Algorithms for continuous distributing monitoring: A survey. In *First International Workshop on Algorithms and Models for Distributed Event Processing*, pages 1–10, 2011.

10 Graham Cormode, Minos N. Garofalakis, S. Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 25–36, 2005.

11 Graham Cormode, S. Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1076–1085, 2008.

12 Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the Twenty-Ninth ACM Symposium on Principles of Database Systems, PODS*, pages 77–86, 2010.

13 Graham Cormode, S. Muthukrishnan, and Wei Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE*, pages 1036–1045, 2007.

14 Graham Cormode and Ke Yi. Tracking distributed aggregates over time-based sliding windows. In *Scientific and Statistical Database Management - 24th International Conference, SSDBM*, pages 416–430, 2012.

15 Abhinandan Das, Sumit Ganguly, Minos N. Garofalakis, and Rajeev Rastogi. Distributed set expression cardinality estimation. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 312–323, 2004.

16 Sutanu Gayen and N. V. Vinodchandran. Algorithms for k-median clustering over distributed streams. In *Computing and Combinatorics - 22nd International Conference, COCOON*, pages 535–546, 2016.

17 Zengfeng Huang, Ke Yi, and Qin Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 295–306, 2012.

18 Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 289–300, 2006.

19 Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 563–574, 2003.

20 Nicolo Rivetti, Yann Busnel, and Achour Mostéfaoui. Efficiently summarizing data streams over sliding windows. In *14th IEEE International Symposium on Network Computing and Applications, NCA*, pages 151–158, 2015.

21 Srikanta Tirthapura and David P. Woodruff. Optimal random sampling from distributed streams revisited. In *Distributed Computing - 25th International Symposium, DISC*, pages 283–297, 2011.

22 David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 941–960, 2012.

Parameterized Aspects of Strong Subgraph Closure

Petr A. Golovach
Department of Informatics, University of Bergen, Norway.
petr.golovach@uib.no

Pinar Heggernes
Department of Informatics, University of Bergen, Norway.
pinar.heggernes@uib.no

Athanasios L. Konstantinidis
Department of Mathematics, University of Ioannina, Greece.
skonstan@cc.uoi.gr

Paloma T. Lima
Department of Informatics, University of Bergen, Norway.
paloma.lima@uib.no

Charis Papadopoulos
Department of Mathematics, University of Ioannina, Greece.
charis@cs.uoi.gr

Abstract

Motivated by the role of triadic closures in social networks, and the importance of finding a maximum subgraph avoiding a fixed pattern, we introduce and initiate the parameterized study of the Strong F-closure problem, where F is a fixed graph. This is a generalization of Strong Triadic Closure, whereas it is a relaxation of F-free Edge Deletion. In Strong F-closure, we want to select a maximum number of edges of the input graph G, and mark them as *strong edges*, in the following way: whenever a subset of the strong edges forms a subgraph isomorphic to F, then the corresponding induced subgraph of G is *not* isomorphic to F. Hence the subgraph of G defined by the strong edges is not necessarily F-free, but whenever it contains a copy of F, there are additional edges in G to destroy that strong copy of F in G.

We study Strong F-closure from a parameterized perspective with various natural parameterizations. Our main focus is on the number k of strong edges as the parameter. We show that the problem is FPT with this parameterization for every fixed graph F, whereas it does not admit a polynomial kernel even when $F = P_3$. In fact, this latter case is equivalent to the Strong Triadic Closure problem, which motivates us to study this problem on input graphs belonging to well known graph classes. We show that Strong Triadic Closure does not admit a polynomial kernel even when the input graph is a split graph, whereas it admits a polynomial kernel when the input graph is planar, and even d-degenerate. Furthermore, on graphs of maximum degree at most 4, we show that Strong Triadic Closure is FPT with the above guarantee parameterization $k - \mu(G)$, where $\mu(G)$ is the maximum matching size of G. We conclude with some results on the parameterization of Strong F-closure by the number of edges of G that are not selected as strong.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Strong triadic closure, Parameterized complexity, Forbidden subgraphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.23

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein; Article No. 23; pp. 23:1–23:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at [14], http://arxiv.org/abs/1802.10386.

Funding This work is supported by Research Council of Norway via project CLASSIS.

1 Introduction

Graph modification problems are at the heart of parameterized algorithms. In particular, the problem of deleting as few edges as possible from a graph so that the remaining graph satisfies a given property has been studied extensively from the viewpoint of both classical and parameterized complexity for the last four decades [23, 11, 8]. For a fixed graph F, a graph G is said to be *F-free* if G has no induced subgraph isomorphic to F. The F-Free Edge Deletion problem asks for the removal of a minimum number of edges from an input graph G so that the remaining graph is F-free. In this paper, we introduce a relaxation of this problem, which we call Strong F-closure. Our problem is also a generalization of the Strong Triadic Closure problem, which asks to select as many edges as possible of a graph as *strong*, so that whenever two strong edges uv and vw share a common endpoint v, the edge uw is also present in the input graph (not necessarily strong). This problem is well studied in the area of social networks [12, 2], and its classical computational complexity has been studied recently both on general graphs and on particular graph classes [22, 18].

In the Strong F-closure problem, we have a fixed graph F, and we are given an input graph G, together with an integer k. The task is to decide whether we can select at least k edges of G and mark them as *strong*, in the following way: whenever the subgraph of G spanned by the strong edges contains an induced subgraph isomorphic to F, then the corresponding induced subgraph of G on the same vertex subset is not isomorphic to F. The remaining edges of G that are not selected as strong, will be called *weak*. Consequently, whenever a subset S of the strong edges form a copy of F, there must be an additional strong or weak edge in G with endpoints among the endpoints of edges in S. A formal definition of the problem is easier to give via spanning subgraphs. If two graphs H and F are isomorphic then we write $H \simeq F$, and if they are not isomorphic then we write $H \not\simeq F$. Given a graph G and a fixed graph F, we say that a (not necessarily induced) subgraph H of G *satisfies the F-closure* if, for every $S \subseteq V(H)$ with $H[S] \simeq F$, we have that $G[S] \not\simeq F$. In this case, the edges of H form exactly the set of strong edges of G.

Strong F-closure

Input: A graph G and a nonnegative integer k.

Task: Decide whether G has a spanning subgraph H that satisfies the F-closure, such that $|E(H)| \geq k$.

Based on this definition and the above explanation, the terms "marking an edge as weak (in G)" and "removing an edge (of G to obtain H)" are equivalent, and we will use them interchangeably. An induced path on three vertices is denoted by P_3. Relating Strong F-closure to the already mentioned problems, observe that Strong P_3-closure is exactly Strong Triadic Closure. Observe also that a solution for F-free Edge Deletion is a solution for Strong F-closure, since the removed edges in the first problem can simply be taken as the weak edges in the second problem. However it is important to note that the reverse is not always true. All of the mentioned problems are known to be NP-hard. The parameterized complexity of F-free Edge Deletion has been studied extensively when parameterized by ℓ, the number of removed edges. With this parameter, the problem is FPT if F is of constant size [4], whereas it becomes W[1]-hard when parameterized by the

size of F even for $\ell = 0$ [15]. Moreover, there exists a small graph F on seven vertices for which F-free Edge Deletion does not admit a polynomial kernel [19] when the problem is parameterized by ℓ. To our knowledge, Strong Triadic Closure has not been studied with respect to parameterized complexity before our work.

In this paper, we study the parameterized complexity of Strong F-closure with three different natural parameters: the number of strong edges, the number of strong edges above guarantee (maximum matching size), and the number of weak edges.

- In Section 3, we show that Strong F-closure is FPT when parameterized by $k = |E(H)|$ for a fixed F. Moreover, we prove that the problem is FPT even when we allow the size of F to be a parameter, that is, if we parameterize the problem by $k + |V(F)|$, except if F has at most one edge. In the latter case Strong F-closure is W[1]-hard when parameterized by $|V(F)|$ even if $k \leq 1$. We also observe that Strong F-closure parameterized by $k + |V(F)|$ admits a polynomial kernel if F has a component with at least three vertices and the input graph is restricted to be d-degenerate. This result is tight in the sense that it cannot be generalized to nowhere dense graphs.
- In Section 4, we focus on the case $F = P_3$, that is, we investigate the parameterized complexity of Strong Triadic Closure. We complement the FPT results of the previous section by proving that Strong Triadic Closure does not admit a polynomial kernel even on split graphs. It is straightforward to see that if F has a connected component on at least three vertices, then a matching in G gives a feasible solution for Strong F-closure. Thus the maximum matching size $\mu(G)$ provides a lower bound for the maximum number of edges of H. Consequently, parameterization above this lower bound becomes interesting. Motivated by this, we study Strong F-closure parameterized by $|E(H)| - \mu(G)$. It is known that Strong Triadic Closure can be solved in polynomial time on subcubic graphs, but it is NP-complete on graphs of maximum degree at most d for every $d \geq 4$ [17]. As a first step in the investigation of the parameterization above lower bound, we show that Strong Triadic Closure is FPT on graphs of maximum degree at most 4, parameterized by $|E(H)| - \mu(G)$.
- Finally, in Section 5, we consider Strong F-closure parameterized by $\ell = |E(G)| - |E(H)|$, that is, by the number of weak edges. We show that the problem is FPT and admits a polynomial (bi-)kernel if F is a fixed graph. Notice that, contrary to the parameterization by $k + |V(F)|$, we cannot hope for FPT results when the problem is parameterized by $\ell + |V(F)|$. This is because, when $\ell = 0$, Strong F-closure is equivalent to asking whether G is F-free, which is equivalent to solving Induced Subgraph Isomorphism that is well known to be W[1]-hard [11, 15]. We also state some additional results and open problems. Our findings are summarized in Table 1 [1].

2 Preliminaries

All graphs considered here are simple and undirected. We refer to Diestel's classical book [9] for standard graph terminology that is undefined here. Given an input graph G, we use the convention that $n = |V|$ and $m = |E|$. Two vertices u and v are *false twins* if $uv \notin E$ and $N(u) = N(v)$, where $N(u)$ is the neighborhood of u. For a graph F, it is said that G is *F-free* if G has no induced subgraph isomorphic to F. For a positive integer d, G is

[1] Due to space constraints in this extended abstract, some proofs marked with an asterisk (*) were removed, whereas other proofs marked with a plus (+) contain only a sketch of the basic idea; full proofs are given in [14].

Table 1 Summary of our results: parameterized complexity analysis of Strong F-closure.

Parameter	Restriction	Parameterized Complexity	Theorem
$\lvert E(H)\rvert + \lvert V(F)\rvert$	$\lvert E(F)\rvert \leq 1$	W[1]-hard	3, 4
	$\lvert E(F)\rvert \geq 2$	FPT	9
	F has a component with ≥ 3 vertices, G is d-degenerate	polynomial kernel	11
$\lvert E(H)\rvert$	F has no isolated vertices	FPT	10
	$F = P_3$, G is split	no polynomial kernel	12
$\lvert E(H)\rvert - \mu(G)$	$F = P_3$, $\Delta(G) \leq 4$	FPT	13
$\lvert E(G)\rvert - \lvert E(H)\rvert$	None	FPT	14
		polynomial (bi-)kernel	15

d-degenerate if every subgraph of G has a vertex of degree at most d. The maximum degree of G is denoted by $\Delta(G)$. We denote by $G + H$ the disjoint union of two graphs G and H. For a positive integer p, pG denotes the disjoint union of p copies of G. A *matching* in G is a set of edges having no common endpoint. The *maximum matching number*, denoted by $\mu(G)$, is the maximum number of edges in any matching of G. We say that a vertex v is *covered* by a matching M if v is incident to an edge of M. An *induced matching*, denoted by qK_2, is a matching M of q edges such that $G[V(M)]$ is isomorphic to qK_2.

Let us give a couple of observations on the nature of our problem. An F-graph of a subgraph H of G is an induced subgraph $H[S] \simeq F$ such that $G[S] \simeq F$. Clearly, if H is a solution for Strong F-closure on G, then there is no F-graph in H, even though H might have induced subgraphs isomorphic to F. For F-free Edge Deletion, note that the removal of an edge that belongs to a forbidden subgraph might generate a new forbidden subgraph. However, for Strong F-closure problem, it is not difficult to see that the removal of an edge that belongs to an F-graph cannot create a new critical subgraph.

▶ **Observation 1.** *Let G be a graph, and let H and H' be spanning subgraphs of G such that $E(H') \subseteq E(H)$. If H satisfies the F-closure for some F, then H' satisfies the F-closure.*

In particular, Observation 1 immediately implies that if an instance of Strong F-closure has a solution, it has a solution with *exactly* k edges.

We conclude this section with some definitions from parameterized complexity and kernelization. A problem with input size n and parameter k is *fixed parameter tractable* (FPT), if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f. A *bi-kernelization* [1] (or *generalized kernelization* [3]) for a parameterized problem P is a polynomial algorithm that maps each instance (x, k) of P with the input x and the parameter k into to an instance (x', k') of some parameterized problem Q such that i) (x, k) is a yes-instance of P if and only if (x', k') is a yes-instance of Q, ii) the size of x' is bounded by $f(k)$ for a computable function f, and iii) k' is bounded by $g(k)$ for a computable function g. The output (x', k') is called a *bi-kernel* (or *generalized kernel*) of the considered problem. The function f defines the size of a bi-kernel and the *bi-kernel has polynomial size* if the function f is polynomial. If $Q = P$, then bi-kernel is called *kernel*. Note that if Q is in NP and P is NP-complete, then the existence of a polynomial bi-kernel implies that P has a polynomial kernel because there exists a polynomial reduction of Q to P. A *polynomial compression* of a parameterized problem P into a (nonparameterized) problem Q is a polynomial algorithm that takes as

an input an instance (x, k) of P and returns an instance x' of Q such that i) (x, k) is a yes-instance of P if and only if x' is a yes-instance of Q, ii) the size of x' is bounded by $p(k)$ for a polynomial p. For further details on parameterized complexity we refer to [8, 11].

3 Parameterized complexity of Strong F-closure

In this section we give a series of lemmata, which together lead to the conclusion that Strong F-closure is FPT when parameterized by k. Observe that in our definition of the problem, F is a fixed graph of constant size. However, the results of this section allow us to also take the size of F as a parameter, making the results more general. We start by making some observations that will rule out some simple types of graphs as F.

▶ **Observation 2.** *Let p be a positive integer. A graph G has a spanning subgraph H satisfying the pK_1-closure if and only if G is pK_1-free, and if G is pK_1-free, then every spanning subgraph H of G satisfies the pK_1-closure.*

By combining Observation 2 and the well known result that Independent Set is W[1]-hard when parameterized by the size of the independent set [11], we obtain the following:

▶ **Proposition 3.** *For a positive integer p,* Strong pK_1-closure *can be solved in time $n^{O(p)}$, and it is co-W[1]-hard for $k \geq 0$ when parameterized by p.*

Using Proposition 3, we assume throughout the remaining parts of the paper that every considered graph F has at least one edge. We have another special case $F = pK_1 + K_2$.

▶ **Proposition 4 (*).** *For a nonnegative integer p,* Strong $(pK_1 + K_2)$-closure *can be solved in time $n^{O(p)}$, and it is co-W[1]-hard for $k \geq 1$ when parameterized by p.*

From now on we assume that $F \neq pK_1$ and $F \neq pK_1 + K_2$. We show that Strong F-closure is FPT when parameterized by k and $|V(F)|$ in this case. We will consider separately the case when F has a connected component with at least 3 vertices and the case $F = pK_1 + qK_2$ for $p \geq 0$ and $q \geq 2$.

▶ **Lemma 5.** *Let F be a graph that has a connected component with at least 3 vertices. Then* Strong F-closure *can be solved in time $2^{O(k^2)}(|V(F)| + k)^{O(k)} + n^{O(1)}$.*

Proof. We show the claim by proving that the problem has a kernel with at most $2^{2k-2}(|V(F)| + k) + 2k - 2$ vertices. Let (G, k) be an instance of Strong F-closure. We recursively apply the following reduction rule in G:

▶ **Rule 5.1.** *If there are at least $|V(F)| + k + 1$ false twins in G, then remove one of them.*

To show that the rule is sound, let $v_1, \ldots, v_p$ be false twins of G for $p = |V(F)| + k + 1$ and assume that G' is obtained from G by deleting v_p. We claim that (G, k) is a yes-instance of Strong F-closure if and only if (G', k) is a yes-instance.

Let (G, k) be a yes-instance. By Observation 1, there is a solution H for (G, k) such that $|E(H)| = k$. Since $|E(H)| = k$, there is $i \in \{1, \ldots, p\}$ such that v_i is an isolated vertex of H. Since $v_1, \ldots, v_p$ are false twins we can assume without loss of generality that $i = p$. Then $H' = H - v_p$ is a solution for (G', k), that is, this is a yes-instance. Assume that (G', k) is a yes-instance of Strong F-closure. Let H' be a solution for the instance with k edges. Denote by H the spanning subgraph of G with $E(H) = E(H')$. We show that H satisfies the F-closure with respect to G. To obtain a contradiction, assume that there is a set of vertices S of G such that $H[S] \simeq F$ and $G[S] \simeq F$. Since H' satisfies the

F-closure with respect to G, $v_p \in S$. Note that v_p is an isolated vertex of H. Because $p = |V(F)| + k + 1$, there is $i \in \{1, \ldots, p-1\}$ such that v_i is an isolated vertex of H and $v_i \notin S$. Let $S' = (S \setminus \{v_p\}) \cup \{v_i\}$. Since v_i and v_p are false twins, $H[S'] = H'[S'] \simeq F$ and $G[S'] \simeq F$; a contradiction. Therefore, we conclude that H satisfies the F-closure with respect to G, that is, H is a solution for (G, k).

It is straightforward to see that the rule can be applied in polynomial time. To simplify notations, assume that (G, k) is the instance of Strong F-closure obtained by the exhaustive application of Rule 5.1. We greedily find an inclusion maximal matching M in G. Notice that the spanning subgraph H of G with $E(H) = M$ satisfies the F-closure because every component of H has at most two vertices and by the assumption of the lemma F has a component with at least 3 vertices. Therefore, if $|M| \geq k$, we have that H is a solution for the instance. Respectively, we return H and stop.

Assume that $|M| \leq k-1$. Let X be the set of end-vertices of the edges of M. Clearly, $|X| \leq 2k-2$ and X is a vertex cover of G. Let $Y = V(G) \setminus X$. We have that Y is an independent set. Every vertex in Y has its neighbors in X. Hence, there are at most $2^{|X|}$ vertices of Y with pairwise distinct neighborhoods. Hence, the vertices of Y can be partitioned into at most $2^{|X|}$ classes of false twins. After applying Rule 5.1, each class of false twins has at most $|V(F)| + k$ vertices. It follows that $|Y| \leq 2^{|X|}(|V(F)| + k)$ and

$$|V(G)| = |X| + |Y| \leq |X| + 2^{|X|}(|V(F)| + k) \leq (2k-2) + 2^{2k-2}(|V(F)| + k).$$

Now we can find a solution for (G, k) by brute force checking all subsets of edges of size k by Observation 1. This can be done it time $|V(G)|^{O(k)}$. Hence, the total running time is $2^{O(k^2)}(|V(F)| + k)^{O(k)} + n^{O(1)}$. ◀

Now we consider the case $F = pK_1 + qK_2$ for $p \geq 0$ and $q \geq 2$. First, we explain how to solve Strong qK_2-closure for $q \geq 2$. We use the random separation technique proposed by Cai, Chen and Chan [6] (see also [8]). To avoid dealing with randomized algorithms and subsequent standard derandomization we use the following lemma stated in [7].

▶ **Lemma 6** ([7]). *Given a set U of size n and integers $0 \leq a, b \leq n$, one can construct in time $2^{O(\min\{a,b\}\log(a+b))} \cdot n \log n$ a family $\mathcal{S}$ of at most $2^{O(\min\{a,b\}\log(a+b))} \cdot \log n$ subsets of U such that the following holds: for any sets $A, B \subseteq U$, $A \cap B = \emptyset$, $|A| \leq a$, $|B| \leq b$, there exists a set $S \in \mathcal{S}$ with $A \subseteq S$ and $B \cap S = \emptyset$.*

▶ **Lemma 7.** *For $q \geq 2$,* Strong qK_2-closure *can be solved in time $2^{O(k \log k)} \cdot n^{O(1)}$.*

Proof. Let (G, k) be an instance of Strong qK_2-closure. If $k < q$, then every spanning subgraph H of G with k edges satisfies the F-closure, that is, (G, k) is a yes-instance of Strong F-closure if $k \leq |E(G)|$. Assume from now that $q \leq k$.

Suppose that G has a vertex v of degree at least k. Let X be the set of edges of G incident to v and consider the spanning subgraph H of G with $E(H) = X$. Since $F = qK_2$ and $q \geq 2$, H satisfies the F-closure. Hence, H is a solution for (G, k). We assume that this is not the case and $\Delta(G) \leq k-1$.

Suppose that (G, k) is a yes-instance. Then by Observation 1, there is a solution H with exactly k edges. Let $A = E(H)$ and denote by X the set of end-vertices of the edges of A. Denote by B the set of edges of $E(G) \setminus A$ that have at least one end-vertex in $N[X]$. Clearly, $A \cap B = \emptyset$. We have that $|A| = k$ and because the maximum degree of G is at most $k-1$, $|B| \leq 2k(k-1)(k-2)$. Applying Lemma 6 for the universe $U = E(G)$, $a = k$ and $b = 2k(k-1)(k-2)$, we construct in time $2^{O(k \log k)} \cdot n^{O(1)}$ a family $\mathcal{S}$ of at most $2^{O(k \log k)} \cdot \log n$ subsets of $E(G)$ such that there exists a set $S \in \mathcal{S}$ with $A \subseteq S$ and $B \cap S = \emptyset$.

For every $S \in \mathcal{S}$, we find (if it exists) a spanning subgraph H of G with k edges such that (i) $E(H) \subseteq S$ and (ii) for every $e_1, e_2 \in S$ that are adjacent or have adjacent end-vertices, it holds that either $e_1, e_2 \in E(H)$ or $e_1, e_2 \notin E(H)$. By Lemma 6, we have that if (G, k) is a yes-instance of Strong F-closure, then it has a solution satisfying (i) and (ii). Hence, if we find a solution for some $S \in \mathcal{S}$, we return it and stop and, otherwise, if there is no solution satisfying (i) and (ii) for some $S \in \mathcal{S}$, we conclude that (G, k) is a no-instance.

Assume that $S \in \mathcal{S}$ is given. We describe the algorithm for finding a solution H with k edges satisfying (i) and (ii). Let R be the set of end-vertices of the edges of S. Consider the graph $G[R]$ and denote by $C_1, \ldots, C_r$ its components. Let $A_i = E(C_i) \cap S$ for $i \in \{1, \ldots, r\}$.

Observe that if H is a solution with k edges satisfying (i) and (ii), then for each $i \in \{1, \ldots, r\}$, either $A_i \subseteq E(H)$ or $A_i \cap E(H) = \emptyset$. It means that we are looking for a solution H such that $E(H)$ is union of some sets A_i, that is, $E(H) = \cup_{i \in I} A_i$ for $I \subseteq \{1, \ldots, r\}$. Let $c_i = |A_i|$ for $i \in \{1, \ldots, r\}$. Clearly, we should have that $\sum_{i \in I} c_i = k$. In particular, it means that if $|A_i| > k$, then the edges of A_i are not in any solution. Therefore, we discard such sets and assume from now that $|A_i| \leq k$ for $i \in \{1, \ldots, r\}$. For $i \in \{1, \ldots, r\}$, denote by w_i the maximum number of edges in A_i that form an induced matching in C_i. Since each $|A_i| \leq k$, the values of w_i can be computed in time $2^k \cdot n^{O(1)}$ by brute force. Observe that for distinct $i, j \in \{1, \ldots, r\}$, the vertices of C_i and C_j are at distance at least two in G and, therefore, the end-vertices of edges of A_i and A_j are not adjacent. It follows, that the problem of finding a solution H is equivalent to the following problem: find $I \subseteq \{1, \ldots, r\}$ such that $\sum_{i \in I} c_i = k$ and $\sum_{i \in I} w_i \leq q$. It is easy to see that we obtain an instance of a variant of the well known Knapsack problem (see, e.g., [16]); the only difference is that we demand $\sum_{i \in I} c_i = k$ instead of $\sum_{i \in I} c_i \geq k$ as in the standard version. This problem can be solved by the standard dynamic programming algorithm (again see, e.g., [16]) in time $O(kn)$.

Since the family $\mathcal{S}$ is constructed in time $2^{O(k \log k)} \cdot n^{O(1)}$ and we consider $2^{O(k \log k)} \cdot \log n$ sets S, we obtain that the total running time is $2^{O(k \log k)} \cdot n^{O(1)}$. ◀

We use Lemma 7 to solve Strong $(pK_1 + qK_2)$-closure.

▶ **Lemma 8 (*).** *For $p \geq 0$ and $q \geq 2$,* Strong $(pK_1 + qK_2)$-closure *can be solved in time $2^{O((k+p) \log(k+p))} \cdot n^{O(1)}$.*

Combining Lemmata 5, 7, and 8, we obtain the following theorem.

▶ **Theorem 9.** *If $F \neq pK_1$ for $p \geq 1$ and $F \neq pK_1 + K_2$ for $p \geq 0$, then* Strong F-closure *is FPT when parameterized by $|V(F)| + k$.*

Notice that if $|E(F)| > k$, then (G, k) is a yes-instance of Strong F-closure. This immediately implies the following corollary.

▶ **Corollary 10.** *If F has no isolated vertices, then* Strong F-closure *is FPT when parameterized by k, even when F is given as a part of the input.*

We conclude this section with a kernel result. Observe that if the input graph G is restricted to be a graph from a sparse graph class $\mathcal{C}$, namely if $\mathcal{C}$ is *nowhere dense* (see [21]) and is closed under taking subgraphs, then the kernel constructed in Lemma 5 becomes polynomial. This observation is based on the results Eickmeyer et al. [13] that allow to bound the number of distinct neighborhoods of vertices in $V(G) \setminus X$ in the construction of the kernel in the proof of Lemma 5. For simplicity, we demonstrate it here on d-degenerate graphs [2].

[2] NP-completeness result for $F = P_3$ restricted to planar graphs (and, thus, 5-degenerate graphs) is given in Section 5.

▶ **Proposition 11** (*). *If F has a connected component with at least 3 vertices, then* Strong F-closure *has a kernel with $k^{O(d)}d(|V(F)|+k)$ vertices on d-degenerate graphs.*

In particular, we have a polynomial kernel when $F = P_3$. Similar results can be obtained for some classes of dense graphs. For example, if G is dK_1-free, then $V(G) \setminus X$ has at most $d-1$ vertices and we obtain a kernel with $2k+d-3$ vertices.

4 Parameterized complexity of Strong Triadic Closure

In this section we study the parameterized complexity of Strong P_3-closure, which is more famously known as Strong Triadic Closure.

Note that Strong Triadic Closure is FPT and admits an algorithm with running time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ by Lemma 5. We complement this result by showing that Strong Triadic Closure does not admit a polynomial kernel, even when the input graph is a split graph. A graph is a *split graph* if its vertex set can be partitioned into an independent set and a clique. Strong Triadic Closure is known to be NP-hard on split graphs [18].

▶ **Theorem 12** (+). Strong Triadic Closure *has no polynomial compression unless NP $\subseteq$ coNP/ poly, even when the input graph is a split graph.*

Proof. We give a reduction from the Set Packing problem: given a universe $\mathcal{U}$ of t elements and subsets $B_1, \ldots, B_p$ of $\mathcal{U}$ decide whether there are at least k subsets which are pairwise disjoint. Set Packing (also known as Rank Disjoint Set problem), parameterized by $|\mathcal{U}|$, does not admit a polynomial compression [10]. Given an instance $(\mathcal{U}, B_1, \ldots, B_p, k)$ for the Set Packing, we construct a split graph G with a clique $U \cup Y$ and an independent set $W \cup X$ as follows:

- The vertices of U correspond to the elements of $\mathcal{U}$.
- For every B_i there is a vertex $w_i \in W$ that is adjacent to all the vertices of $(U \cup Y) \setminus B_i$.
- X and Y contain additional $2t$ vertices with $X = \{x_1, \ldots, x_t\}$ and $Y = \{y_1, \ldots, y_t\}$ such that y_i is adjacent to all the vertices of $(W \cup X) \setminus \{x_i\}$ and x_i is adjacent to all the vertices of $(U \cup Y) \setminus \{y_i\}$.

Notice that the clique of G contains $2t$ vertices. We will show that there are at least k pairwise disjoint sets in $\{B_1, \ldots, B_p\}$ if and only if there is a solution for Strong P_3-closure on G with at least $k' = |E(U \cup Y)| + (k+t)/2$ edges.

Assume that $\mathcal{B}'$ is a family of k pairwise disjoint sets of $B_1, \ldots, B_p$. For every $B'_i \in \mathcal{B}'$ we choose three vertices w_i, y_i, x_i from W, Y, and X, respectively, such that x_i is non-adjacent to y_i with the following strong edges: w_i is strongly adjacent to y_i and x_i is strongly adjacent to the vertices of B'_i in U. We also make weak the edges inside the clique between the vertices of B'_i and y_i. All other edges incident to w_i and x_i are weak. Let W', Y', X' be the set of vertices that are chosen from the family $\mathcal{B}'$ according to the previous description. Every vertex of $W \setminus W'$ is not incident to a strong edge. For the $t-k$ vertices of $Y \setminus Y'$ we choose a matching and for each matched pair $y_j, y_{j'}$ we make the following edges strong: $x_j y_{j'}$ and $x_{j'} y_j$ where x_j and $x_{j'}$ are non-adjacent to y_j and $y_{j'}$, respectively. Moreover each edge $y_j y_{j'}$ of the clique is weak and all other edges incident to x_j and $x_{j'}$ are weak. The rest of the edges inside the clique $U \cup Y$ are strong. It is not difficult to verify that the described labeling satisfies the P_3-closure with the claimed number of strong edges.

For the opposite direction, assume that H is a subgraph of G that satisfies the P_3-closure with at least k' edges. For a vertex $v \in W \cup X$, let $S(v)$ be the strong neighbors of v in H and let $B(v)$ be the non-neighbors of v in $U \cup Y$. Our task is to show that for any two vertices u, v of $W \cup X$ with non-empty sets $S(u), S(v)$, we have $B(u) \cap B(v) = \emptyset$. We accomplish

that, by showing the following arguments: (i) for any weak edge e inside the clique there must be strong edges between the endpoints of e and *special* vertices of the independent set, (ii) in order to achieve the bound k', there are strong edges incident to the vertices of W, (iii) any component of the clique spanned by weak edges induces a tree of height one, and (iv) for any two vertices u, v of $W \cup X$ with non-empty sets $S(u), S(v)$, their non-neighborhoods $B(u), B(v)$ do not have the containment property. Then by the last two arguments we know that all vertices of W that are incident to at least one strong edge in H must have disjoint non-neighborhood. Since $B(w_i) = B_i$, there are k pairwise disjoint sets in $\{B_1, \ldots, B_p\}$ for the k vertices of W that are incident to at least one strong edge in H. Therefore there is a solution for the Set Packing problem for $(\mathcal{U}, B_1, \ldots, B_p, k)$. ◀

Let F be a graph that has at least one component with at least three vertices. If M is a matching in a graph G, then the spanning subgraph H of G with $E(H) = M$ satisfies the F-closure. It implies that an instance (G, k) of Strong F-Closure is a yes-instance of the problem if the maximum matching size $\mu(G) \geq k$. Since a maximum matching can be found in polynomial time [20], we can solve Strong F-Closure in polynomial time for such instances. This gives rise to the question about the parameterized complexity of Strong F-Closure with the parameter $r = k - \mu(G)$. We show that Strong Triadic Closure is FPT with this parameter for the instances where $\Delta(G) \leq 4$. Note that Strong Triadic Closure is NP-complete on graphs G with $\Delta(G) \leq d$ for every $d \geq 4$ [17].

▶ **Theorem 13** (+). Strong Triadic Closure *can be solved in time* $2^{O(r)} \cdot n^{O(1)}$ *on graphs of maximum degree at most 4, where* $r = k - \mu(G)$.

Proof. Let (G, k) be an instance of Strong Triadic Closure such that $\Delta(G) \leq 4$. We construct the set of vertices X and the set of edges A as follows. Initially, $X = \emptyset$ and $A = \emptyset$. Then we exhaustively perform the following steps in a greedy way:

1. If there exists a copy of K_4 in $G - X$, we add the vertices of this K_4 to X and the edges between these vertices to A.
2. If there exists a triangle T in $G - X$ such that $\mu(G - X) < 3 + \mu(G - X - T)$, we add the vertices of T to X and and the edges of T to A.

Let M be a maximum matching of $G - X$ for the obtained set X. Note that the spanning subgraph H of G with the set of edges $A \cup M$ satisfies the P_3-closure. Assume that Step 1 was applied p times and we used Step 2 q times. Clearly, $|A| = 6p + 3q$. Notice that the vertices of a copy of K_4 can be incident to at most 4 edges of a matching and the complete graph with 4 vertices has 6 edges. Observe also that by Step 2, we increase the size of A by 3 and $\mu(G - X) - \mu(G - X - T) \leq 2$. This implies that $|E(H)| = |A| + |M| \geq \mu(G) + 2p + q$. Therefore, if $2p + q \geq r$, (G, k) is a yes-instance. Assume from now that this is not the case. In particular $|X| \leq 4r$ and $G' = G - X$ is a K_4-free graph.

We need some structural properties of G' and (possible) solutions for the considered instance. By Step 2, we know that for every triangle T in G': (i) T contains no edge of M and (ii) every vertex of T is incident to an edge of M. We say that a solution H for (G, k) is *regular* if $H - X$ is a disjoint union of triangles, edges and isolated vertices. We also say that a solution H is *triangle-maximal* if (i) it contains the maximum number of edges and, subject to (i), (ii) contain the maximum number of pairwise distinct triangles. By the fact that $\Delta(G) \leq 4$, it can be proved that if (G, k) is a yes-instance, then it has a triangle-maximal regular solution. Next we derive the following properties for triangles in G' that are at distance one or more from X.

- For any triangle T at distance one from X, if T is included in H then H contains no other edge incident to T.

- For any triangle T at distance at least two from X that does not intersect any other triangle, T is included in every triangle-maximal regular solution for (G, k).
- If T_1 and T_2 are two intersecting triangles in G', then (i) T_1 and T_2 have one edge in common and (ii) no other triangle intersects T_1 or T_2.
- If T_1 and T_2 are two intersecting triangles such that T_1 is at distance at least two from X, then either T_1 or T_2 is included in every regular triangle-maximal solution for (G, k).

Now we are ready to solve the problem by finding a triangle-maximal regular solution if it exists. The crucial step is to sort out triangles in G'. Since $|X| \leq 4r$ and since every vertex of X has at least two neighbors inside X, we have $|N_G(X)| \leq 8r$. By the triangle properties, at most 2 triangles of G' contain the same vertex. Thus, the number of pairwise distinct triangles in G' that are at distance at most one from X is at most $16r$. We list all these triangles, and branch on all at most 2^{16r} choices of the triangles that are included in a triangle-maximal regular solution. Then, for each choice of these triangles, we try to extend the partial solution. If we obtain a solution for one of the choices we return it; otherwise, the algorithm returns NO.

Assume that we are given a set $\mathcal{T}_1$ of triangles at distance one from X that should be in a solution. We apply the following reduction rule.

▶ **Rule 13.1.** *Set $G = G - \cup_{T\in\mathcal{T}_1} T$ and set $k = k - 3|\mathcal{T}_1|$.*

Now we deal with triangles that are at distance at least 2. Consider the set $\mathcal{T}_2$ of triangles in G' that are at distance at least 2 from X and have no common vertices with other triangles in G'. Such triangles are in every triangle-maximal regular solution which gives us the following:

▶ **Rule 13.2.** *Set $G = G - \cup_{T\in\mathcal{T}_2} T$ and set $k = k - 3|\mathcal{T}_2|$.*

To consider the remaining triangles for every such a triangle T, T is intersecting with a unique triangle T' of G' and T, T' are sharing an edge. Let $\mathcal{T}_3$ be the set of triangles in G' that are at distance at least 2 from X in G and have a common edge with a triangle at distance one from X.

▶ **Rule 13.3.** *Set $G = G - \cup_{T\in\mathcal{T}_3} T$ and set $k = k - 3|\mathcal{T}_3|$.*

Let $G' = G - X$. The remaining triangles in G' at distance at least 2 from X in G form pairs $\{T_1, T_2\}$ such that T_1 and T_2 have a common edge and are not intersecting any other triangle. Let $\mathcal{P}$ be the set of all such pairs. We apply the property that a triangle-maximal regular solution contains either T_1 or T_2 to construct the following rule.

▶ **Rule 13.4.** *For every pair $\{T_1, T_2\} \in \mathcal{P}$, delete the vertices of T_1 and T_2 from G, construct a new vertex u and make it adjacent to the vertices of $N_G((T_1 \setminus T_2) \cup (T_2 \setminus T_1))$. Set $k = k - 3|\mathcal{P}|$.*

Denote by $(\hat{G}, \hat{k})$ the instance obtained from (G, k) by the application of Rule 13.4. We can show the following important claim.

▶ **Claim 13.1.** *The instance (G, k) has a regular solution H that has no triangles in $G - X$ at distance one from X if and only if there is a solution $\hat{H}$ for $(\hat{G}, \hat{k})$ such that $\hat{H} - X$ is a disjoint union of edges and isolated vertices.*

Thus we have to find a solution for the instance $(\hat{G}, \hat{k})$ such that $\hat{H} - X$ is a disjoint union of edges and isolated vertices. We do it by branching on all possible choices of edges in a solution that are incident to the vertices of X. Since $|X| \leq 4r$ and $\Delta(G) \leq 4$, there are at most $16r$ edges that are incident to the vertices of X and, therefore, we branch on at most

2^{16r} choices of a set of edges S. Then for each choice of S, we are trying to extend it to a solution. If we can do it for one of the choices, we return the corresponding solution, and the algorithm returns NO otherwise. First, we verify whether the spanning subgraph of G with the set of edges S satisfies the P_3-closure. If it is not so, we discard the current choice of S since, trivially, S cannot be extended to a solution. Assume that this is not the case. Let $R = \hat{G} - X$. We modify R by the exhaustive application of the following rule.

▶ **Rule 13.5.** *If there is $xy \in E(R)$ such that there is $z \in X$ with $xz \in S$ and $yz \notin E(\hat{G})$, then delete xy from R.*

Let R' be the graph obtained from R by the rule. Observe that the edges deleted by Rule 13.5 cannot belong to a solution. Hence, to extend S, we have to complement it by some edges of R' that form a matching. Every matching of R' could be used to complement S. Respectively, we find a maximum matching M in R' in polynomial time. Then the spanning subgraph $\hat{H}$ of $\hat{G}$ with $E(\hat{H}) = S \cup M$ satisfies the P_3-closure. We verify whether $|S| + |M| \geq \hat{k}$. If it holds, we return $\hat{H}$. Otherwise, we discard the current choice of S. ◀

5 Concluding remarks

To complement our results so far, we give here the parameterized complexity results when our problem is parameterized by the number of weak edges. The following result is not difficult to deduce using similar ideas to those used in proving that F-free Edge Deletion is FPT by the number of deleted edges [4].

▶ **Theorem 14 (*).** *For every fixed graph F,* Strong F-closure *can be solved in time $2^{O(\ell)} \cdot n^{O(1)}$, where $\ell = |E(G)| - k$.*

Next we show that Strong F-closure has a polynomial bi-kernel with this parameterization whenever F is a fixed graph. We obtain this result by constructing bi-kernelization that reduces Strong F-closure to the d-Hitting Set problem that is the variant of Hitting Set with all the sets in $\mathcal{C}$ having d elements. Notice that this result comes in contrast to the F-free Edge Deletion problem, as it is known that there are fixed graphs F for which there is no polynomial compression [5] unless NP $\subseteq$ coNP/ poly.

▶ **Theorem 15 (*).** *For every fixed graph F,* Strong F-closure *has a polynomial bi-kernel, when parameterized by $\ell = |E(G)| - k$.*

We would like to underline that Theorems 14 and 15 are fulfilled for the case when F is a fixed graph of constant size, as the degree of the polynomial in the running time of our algorithm depends on the size of F and, similarly, the size of F is in the exponent of the function defining the size of our bi-kernel. We can hardly avoid this dependence as it can be observed that for $\ell = 0$, Strong F-closure is equivalent to asking whether the input graph G is F-free, that is, we have to solve the Induced Subgraph Isomorphism problem. It is well known that Induced Subgraph Isomorphism parameterized by the size of F is W[1]-hard when F is a complete graph or graph without edges [11], and the problem is W[1]-hard when F belongs to other restricted families of graphs [15].

We conclude with a few open problems. An interesting question is whether Strong Triadic Closure is FPT when parameterized by $r = k - \mu(G)$. We proved that this holds on graphs of maximum degree at most 4, and we believe that this question is interesting not only on general graphs but also on various other graph classes. In particular, what can be said about planar graphs? To set the background, we show that Strong Triadic Closure is NP-hard on this class.

▶ **Theorem 16 (*).** Strong Triadic Closure *is NP-hard on planar graphs.*

The same question can be asked for the case when $F \neq P_3$ has a connected component with at least three vertices. As a first step, we give an FPT result when F is a star.

▶ **Theorem 17 (*).** *For every $t \geq 3$,* Strong $K_{1,t}$-closure *can be solved in time $2^{O(r^2)} \cdot n^{\mathcal{O}(1)}$, where $r = k - \mu(G)$.*

Another direction of research is to extend Strong F-closure by replacing F with a list of forbidden subgraphs $\mathcal{F}$ and settle the complexity differences compared to $\mathcal{F} = \{F\}$.

References

1 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving max-r-sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

2 L. Backstrom and J. Kleinberg. Romantic partnerships and the dispersion of social ties: a network analysis of relationship status on facebook. In *CSCW 2014*, pages 831–841, 2014.

3 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

4 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.

5 L. Cai and Y. Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71:731–757, 2015.

6 L. Cai, S.M. Chan, and S.O. Chan. Random separation: a new method for solving fixed-cardinality optimization problems. In *IWPEC 2006*, pages 239–250, 2006.

7 R. Chitnis, M. Cygan, M. Hajiaghayi, M. Pilipczuk, and M. Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016.

8 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

10 M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and ids. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014.

11 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science, Springer, 1997.

12 D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

13 K. Eickmeyer, A. C. Giannopoulou, S. Kreutzer, O. Kwon, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *ICALP 2017*, pages 63:1–63:14, 2017.

14 P.A. Golovach, P. Heggernes, A.L. Konstantinidis, P.T. Lima, and C. Papadopoulos. Parameterized aspects of strong subgraph closure. *Available on arxiv.org/abs/1802.10386*, 2018.

15 S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289(2):997–1008, 2002.

16 J. M. Kleinberg and É. Tardos. *Algorithm design*. Addison-Wesley, 2006.

17 A. L. Konstantinidis, S. D. Nikolopoulos, and C. Papadopoulos. Strong triadic closure in cographs and graphs of low maximum degree. In *COCOON 2017*, pages 346–358, 2017.

18 A. L. Konstantinidis and C. Papadopoulos. Maximizing the strong triadic closure in split graphs and proper interval graphs. In *ISAAC 2017*, pages 53:1–53:12, 2017.

19 S. Kratsch and M. Wahlstrom. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10:193–199, 2013.

20 S. Micali and V. V. Vazirani. An O(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *FOCS 1980*, pages 17–27, 1980.

21 J. Nesetril and P. Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

22 S. Sintos and P. Tsaparas. Using strong triadic closure to characterize ties in social networks. In *KDD 2014*, pages 1466–1475, 2014.

23 M. Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981.

Parameterized Orientable Deletion

Tesshu Hanaka
Department of Information and System Engineering, Chuo University, Tokyo, Japan
hanaka.91t@g.chuo-u.ac.jp

Ioannis Katsikarelis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243,
LAMSADE, 75016 Paris, France
ioannis.katsikarelis@dauphine.fr

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243,
LAMSADE, 75016 Paris, France
michail.lampis@dauphine.fr

Yota Otachi
Kumamoto University, Kumamoto, 860-8555, Japan
otachi@cs.kumamoto-u.ac.jp

Florian Sikora
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243,
LAMSADE, 75016 Paris, France
florian.sikora@dauphine.fr

Abstract

A graph is d-orientable if its edges can be oriented so that the maximum in-degree of the resulting digraph is at most d. d-orientability is a well-studied concept with close connections to fundamental graph-theoretic notions and applications as a load balancing problem. In this paper we consider the d-Orientable Deletion problem: given a graph $G = (V, E)$, delete the minimum number of vertices to make G d-orientable. We contribute a number of results that improve the state of the art on this problem. Specifically:

- We show that the problem is W[2]-hard and $\log n$-inapproximable with respect to k, the number of deleted vertices. This closes the gap in the problem's approximability.
- We completely characterize the parameterized complexity of the problem on chordal graphs: it is FPT parameterized by $d + k$, but W-hard for each of the parameters d, k separately.
- We show that, under the SETH, for all d, ϵ, the problem does not admit a $(d + 2 - \epsilon)^{\mathrm{tw}}$, algorithm where tw is the graph's treewidth, resolving as a special case an open problem on the complexity of PseudoForest Deletion.
- We show that the problem is W-hard parameterized by the input graph's clique-width. Complementing this, we provide an algorithm running in time $d^{O(d \cdot \mathrm{cw})}$, showing that the problem is FPT by $d + \mathrm{cw}$, and improving the previously best know algorithm for this case.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph orientations, FPT algorithms, Treewidth, SETH

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.24

Funding This work was financially supported by the "PHC Sakura" program (project GRAPA, number: 38593YJ), implemented by the French Ministry of Foreign Affairs, the French Ministry of Higher Education and Research and the Japan Society for Promotion of Science.

1 Introduction

In this paper we study the following natural optimization problem: we are given a graph $G = (V, E)$ and an integer d, and are asked to give directions to the edges of E so that in the resulting digraph as many vertices as possible have in-degree at most d. Equivalently, we are looking for an orientation of E such that the set of vertices K whose in-degree is strictly more than d is minimized. Such an orientation is called a d-orientation of $G[V \setminus K]$, and we say that K is a set whose deletion makes the graph d-orientable. The problem of orienting the edges of an undirected graph so that the in-degree of all, or most, vertices stays below a given threshold has been extensively studied in the literature, in part because of its numerous applications. In particular, one way to view this problem is as a form of scheduling, or load balancing, where edges represent jobs and vertices represent machines. In this case the in-degree represents the load of a machine in a given assignment, and minimizing it is a natural objective (see e.g. [6, 10, 14, 20]). Finding an orientation where all in- or out-degrees are small is also of interest for the design of efficient data structures [11]. For more applications we refer the reader to [2, 3, 4, 5, 9] and the references therein.

State of the art. d-orientability has been well-studied in the literature, both because of its practical motivations explained above, but also because it is a basic graph property that generalizes and is closely related to fundamental concepts such as d-degeneracy (as a graph is d-degenerate if and only if it admits an *acyclic* d-orientation), and bounded degree. This places d-Orientable Deletion in a general context of graph editing problems that measure the distance of a given graph from having one of these properties [7, 17].

Deciding if an unweighted graph is d-orientable is solvable in polynomial time [5], though the problem becomes APX-hard [14] and even W-hard parameterized by treewidth [19] if one allows edge weights. In this paper we focus on unweighted graphs, for which computing the minimum number of vertices that need to be deleted to make a graph d-orientable is easily seen to be NP-hard, as the case $d = 0$ corresponds to Vertex Cover. This hardness has motivated the study of both polynomial-time approximation and parameterized algorithms, as well as algorithms for specific graph classes. For approximation, if the objective function is to maximize the number of non-deleted vertices, the problem is known to be $n^{1-\epsilon}$-inapproximable; if one seeks to minimize the number of deleted vertices, the problem admits an $O(\log d)$-approximation, but it is not known if this can be improved to a constant [2]. From the parameterized point of view, the problem is W[1]-hard for any fixed d if the parameter is the number of non-deleted vertices [9]. To the best of our knowledge, the complexity of this problem parameterized by the number of deleted vertices is open.

We remark that sometimes in the literature a d-orientation is an orientation where all *out-degrees* are at most d, but this can be seen to be equivalent to our formulation by reversing the direction of all edges. d-Orientable Deletion has sometimes been called Min-$(d+1)$-Heavy/Max-d-Light [2], depending on whether one seeks to minimize the number of deleted vertices, or maximize the number of non-deleted vertices (the two are equivalent in the context of exact algorithms). The problem of finding an orientation minimizing the maximum out-degree has also been called Minimum Maximum Out-degree [5].

An important special case that has recently attracted attention from the FPT algorithms point of view is that of $d = 1$. 1-orientable graphs are called pseudo-forests, as they are exactly the graphs where each component contains at most one cycle. 1-Orientable Deletion, also known as PseudoForest Deletion, has been shown to admit a 3^k algorithm, where k is the number of vertices to be deleted [8, 18].

Our contribution. We study the complexity of d-Orientable Deletion mostly from the point of view of exact FPT algorithms. We contribute a number of new results that improve the state of the art and, in some cases, resolve open problems from the literature.

We first consider the parameterized complexity of the problem with respect to the natural parameter k, the number of vertices to be deleted to make the graph d-orientable. We show that for any fixed $d \geqslant 2$, d-Orientable Deletion is W[2]-hard parameterized by k. This result is tight in two respects: it shows that, under the ETH, the trivial n^k algorithm that tries all possible solutions is essentially optimal; and it cannot be extended to the case $d = 1$, as in this case the problem is FPT [8]. Because our proof is a reduction from Dominating Set that preserves the optimal, we also show that the problem cannot be approximated with a factor better than $\ln n$. This matches the performance of the algorithm given in [2], and closes a gap in the status of this problem, as the previously best known hardness of approximation bound was 1.36 [2].

Second, we consider the complexity of d-Orientable Deletion when restricted to chordal graphs, motivated by the work of [9], who study the problem on classes of graphs with polynomially many minimal separators. We are able to completely characterize the complexity of the problem for this class of graphs with respect to the two main natural parameters d and k: the problem is W[1]-hard parameterized by d, W[2]-hard parameterized by k, but solvable in time roughly $d^{O(d+k)}$, and hence FPT when parameterized by $d + k$. We recall that the problem is poly-time solvable on chordal graphs when d is a constant [9], and trivially in P in general graphs when k is a constant, so these results are in a sense tight.

Third, we consider the complexity of d-Orientable Deletion parameterized by the input graph's treewidth, perhaps the most widely studied graph parameter. Our main contribution here is a lower bound which, assuming the Strong ETH, states that the problem cannot be solved in time less than $(d + 2)^{\mathrm{tw}}$, for any constant $d \geqslant 1$. As a consequence, this shows that the 3^{tw} algorithm given for PseudoForest Deletion in [8] is optimal under the SETH. We recall that Bodlaender et al. [8] had explicitly posed the existence of a better treewidth-based algorithm as an open problem; our results settle this question in the negative, assuming the SETH. Our result also extends the lower bound of [16] which showed that Vertex Cover (which corresponds to $d = 0$) cannot be solved in $(2 - \epsilon)^{\mathrm{tw}}$.

Finally, we consider the complexity of the problem parameterized by clique-width. We recall that clique-width is probably the second most widely studied graph parameter in FPT algorithms (after treewidth), so after having settled the complexity of d-Orientable Deletion with respect to treewidth, investigating clique-width is a natural question. On the positive side, we present a dynamic programming algorithm whose complexity is roughly $d^{O(d \cdot \mathrm{cw})}$, and is therefore FPT when parameterized by $d + \mathrm{cw}$. This significantly improves upon the dynamic programming algorithm for this case given in [9], which runs in time roughly $n^{O(d \cdot \mathrm{cw})}$. The main new idea of this algorithm, leading to its improved performance, is the observation that sufficiently large entries of the DP table can be merged using a more careful characterization of feasible solutions that involve large bi-cliques. On the negative side, we present a reduction showing that d-Orientable Deletion is W[1]-hard if cw is the only parameter. This presents an interesting contrast with the case of treewidth: for both parameters we can obtain algorithms whose running time is a function of d and the width; however, because graphs of treewidth w always admit a w-orientation (since they are w-degenerate), this immediately also shows that the problem is FPT for treewidth, while our results imply that obtaining a similar result for clique-width is impossible (under standard assumptions). Due to space constraints, some proofs (marked with a ★) are omitted.

2 Definitions and Preliminaries

Complexity background. We assume that the reader is familiar with the basic definitions of parameterized complexity, such as the classes FPT and W[1] [13]. We will also make use of the *Exponential Time Hypothesis* (ETH), a conjecture by Impagliazzo et al. asserting that there is no $2^{o(n)}$-time algorithm for 3-SAT on instances with n variables [15]. We also use a corollary (a slightly weaker statement) of the Strong Exponential Time Hypothesis (SETH), stating that SAT cannot be solved in time $O^*((2-\epsilon)^n)$ for any $\epsilon > 0$ [15].

Graph widths. We also make use of standard graph width measures, such as pathwidth, treewidth, and clique-width, denoted as $\mathrm{pw}, \mathrm{tw}, \mathrm{cw}$ respectively. For the definitions we refer the reader to standard textbooks [13, 12]. We recall the following standard relations:

▶ **Lemma 1.** *For all graphs $G = (V, E)$ we have $tw(G) \leqslant pw(G)$ and $cw(G) \leqslant pw(G) + 2$.*

Graphs and Orientability. We use standard graph-theoretic notation. If $G = (V, E)$ is a graph and $S \subseteq V$, $G[S]$ denotes the subgraph of G induced by S. For $v \in V$, the set of neighbors of v in G is denoted by $N_G(v)$, or simply $N(v)$, and $N_G(S) := (\bigcup_{v \in S} N(v)) \setminus S$ will often be written just $N(S)$. We define $N[v] := N(v) \cup \{v\}$ and $N[S] := N(S) \cup S$. Depending on the context, we use (u, v), where $u, v \in V$ to denote either an undirected edge connecting two vertices u, v, or an arc (that is, a directed edge) with tail u and head v. An *orientation* of an undirected graph $G = (V, E)$ is a directed graph on the same set of vertices obtained by replacing each undirected edge $(u, v) \in E$ with either the arc (u, v) or the arc (v, u). In a directed graph we define the in-degree $\delta^-(v)$ of a vertex u as the number of arcs whose head is u. A d-orientation of a graph $G = (V, E)$ is an orientation of G such that all vertices have in-degree at most d. If such an orientation exists, we say that G is d-orientable. Deciding if a given graph is d-orientable is solvable in polynomial time, even if d is part of the input [5]. Let us first make some easy observations on the d-orientability of some basic graphs.

▶ **Lemma 2 (★).** *K_{2d+1}, the clique on $2d+1$ vertices, is d-orientable. Furthermore, in any d-orientation of K_{2d+1} all vertices have in-degree d.*

▶ **Lemma 3 (★).** *The complete bipartite graph $K_{2d+1,2d}$ is not d-orientable.*

▶ **Definition 4.** In d-Orientable Deletion we are given as input a graph $G = (V, E)$ and an integer d. We are asked to determine the smallest set of vertices $K \subseteq V$ (the *deletion set*) such that $G[V \setminus K]$ admits a d-orientation.

▶ **Definition 5.** In Capacitated-d-Orientable Deletion we are given as input a graph $G = (V, E)$, an integer $d \geqslant 1$, and a *capacity* function $\mathbf{c} : V \to \{0, \ldots, d\}$. We are asked to determine the smallest set of vertices $K \subseteq V$ such that $G[V \setminus K]$ admits an orientation with the property that for all $u \in V \setminus K$, the in-degree of u is at most $\mathbf{c}(u)$.

It is clear that Capacitated-d-Orientable Deletion generalizes d-Orientable Deletion, which corresponds to the case where we have $\mathbf{c}(u) = d$ for all vertices. It is, however, not hard to see that the two problems are in fact equivalent, as shown in the following lemma. Furthermore, the following lemma shows that increasing d can only make the problem harder.

▶ **Lemma 6.** *There exists a polynomial-time algorithm which, given an instance $[G = (V, E), d, \mathbf{c}]$ of* Capacitated-d-Orientable Deletion*, and an integer $d' \geqslant d$, produces an equivalent instance $[G' = (V', E'), d']$ of* d-Orientable Deletion*, with the same optimal value and the following properties: $pw(G') \leqslant pw(G) + 2d' + 1$, $cw(G') \leqslant cw(G) + 4$, and if G is chordal then G' is chordal.*

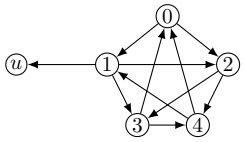


Figure 1 A 2-orientation of a clique K_5. Observe that any edge connecting a vertex u to the clique must be oriented towards u (setting its *capacity*) to maintain a 2-orientation.

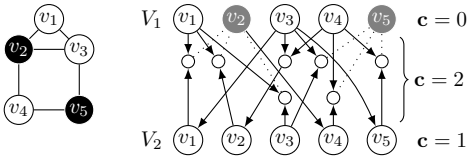


Figure 2 Left: A graph with its dominating set in black. **Right:** The corresponding instance and 2-orientation, where deleted vertices are in gray. Original edges with a deleted vertex as an endpoint are dotted.

Proof (Sketch). The idea of the reduction is to construct for every $u \in V$ for which $\mathbf{c}(u) < d'$ a clique of $K_{2d'+1}$ vertices and connect $d' - \mathbf{c}(u)$ vertices of the clique to u (see Figure 1). Lemma 2 ensures that the edges connecting the clique to u will be oriented towards u, simulating its decreased capacity. We can then argue that in the new instance there always exists an optimal solution that does not delete any of the new vertices, therefore optimal solutions are preserved. ◀

3 Hardness of Approximation and W-hardness

In this section we present a reduction from Dominating Set to d-Orientable Deletion for $d \geqslant 2$ that exactly preserves the size of the solution. As a result, this establishes that, for any fixed $d \geqslant 2$, d-Orientable Deletion is W[2]-hard, and the minimum solution cannot be approximated with a better than logarithmic factor. We observe that it is natural that our reduction only works for $d \geqslant 2$, as the problem is known to be FPT for $d = 1$, which is known as PseudoForest Deletion, and $d = 0$, which is equivalent to Vertex Cover.

▶ **Theorem 7.** *For any $d \geqslant 2$, d-Orientable Deletion is W[2]-hard parameterized by the solution size k. Furthermore, for any $d \geqslant 2$, d-Orientable Deletion cannot be solved in time $f(k) \cdot n^{o(k)}$, unless the ETH is false.*

Proof. We will describe a reduction from Dominating Set, which is well-known to be W[2]-hard and not solvable in $f(k) \cdot n^{o(k)}$ under the ETH, to Capacitated-d-Orientable Deletion for $d = 2$. We will then invoke Lemma 6 to obtain the claimed result for d-Orientable Deletion. Let $[G(V, E), k]$ be an instance of Dominating Set. We begin by constructing a bipartite graph H by taking two copies of V, call them V_1, V_2. For each $v \in V_2$ we construct a binary tree with $|N_G[v]|$ leaves. We identify the root of this binary tree with $v \in V_2$ and its leaves with the corresponding vertices in V_1. We now define the capacities of our vertices: each vertex of V_1 has capacity 0; each internal vertex of the binary trees has capacity 2; and each vertex of V_2 has capacity 1.

We will now claim that G has a dominating set of size k if and only if H can be oriented in a way that respects the capacities by deleting at most k vertices.

For the forward direction, suppose that there is a dominating set in G of size k. In H we delete the corresponding vertices of V_1. We argue that the remaining graph is orientable in a way that respects the capacities. We compute an orientation as follows:

1. We orient the remaining incident edges away from every vertex of V_1 that is not deleted.
2. For each non-leaf vertex u of the binary tree rooted at $v \in V_2$ we define the orientation of the edge connecting u to its parent as follows: u is an ancestor of a set $S_u \subseteq N_G[v]$ of vertices of V_1. If S_u contains a deleted vertex, then we orient the edge connecting u to its parent towards u, otherwise we orient it towards u's parent.

The above description completely defines the orientation of the remaining graph (see also Figure 2). Let us argue why the orientation respects all capacities. This should be clear for vertices of V_1. For any non-leaf vertex u of a binary tree, if we orient the edge connecting it to its parent away from u, then the in-degree of u is at most 2, which is its capacity. On the other hand, if we orient this edge towards u, there is a deleted vertex in S_u. However, this implies either that one of u's children has been deleted, or that one of the edges connecting u to one of its children is oriented away from u. In both cases, the in-degree of u is at most 2, equal to its capacity. Finally, for each $u \in V_2$, if we started with a dominating set, then one of the children of u in the binary tree is either deleted or its edge to u is oriented towards it.

For the converse direction, suppose that there is a set of k vertices in H whose deletion makes the graph orientable in a way that respects the capacities. Suppose now that we have a solution that deletes some vertex $v \in V_2$ or some internal vertex of a binary tree. We re-introduce v in the graph, orient all its incident edges towards v, and then delete one of the children of v. This preserves the size and validity of the solution. Repeating this argument ends with a solution that only deletes vertices of V_1. We now claim that these k vertices are a dominating set. To see this, observe that any undeleted vertex of V_1 has all its edges connecting it to binary trees oriented away from it. Hence, if there is a binary tree with root $v \in V_2$ such that none of its leaves are undeleted, all its internal edges must be oriented towards v, which would make the in-degree of v greater than its capacity. ◀

▶ **Corollary 8 (★).** *For any $d \geqslant 2$, if there exists a polynomial-time $o(\log n)$-approximation for d-Orientable Deletion, then P=NP.*

4 Chordal Graphs

In this section we consider the complexity of d-Orientable Deletion on chordal graphs parameterized by either d or k (the number of deleted vertices). Our main results state that the problem is W-hard for each of these parameters individually (Theorems 9 and 10); however, the problem is FPT parameterized by $d + k$ (Theorem 11).

▶ **Theorem 9 (★).** *d-Orientable Deletion is W[1]-hard on chordal graphs parameterized by d. Furthermore, it cannot be solved in $n^{o(d)}$ under the ETH.*

▶ **Theorem 10.** *d-Orientable Deletion is W[2]-hard on chordal graphs parameterized by the solution size k. Furthermore, under the ETH it cannot be solved in time $n^{o(k)}$.*

Proof. We start from an instance of Dominating Set: we are given a graph $G = (V, E)$ and an integer k and are asked if there exists a dominating set of size k. We will retain the same value of k and construct a chordal instance of Capacitated-d-Orientable Deletion, for which we later invoke Lemma 6. Let $|V| = n$ and we assume without loss of generality that $n - k$ is odd (otherwise we can add an isolated vertex to G). We construct G' as follows. Take two copies of V, call them V_1, V_2 and add all possible edges between vertices of V_2. For each $u \in V$, we connect $u \in V_1$ with all vertices $v \in V_2$ such that $v \in N_G[u]$, i.e. all vertices v that are neighbors of u in G. Let us also define the capacities: each $u \in V_1$ has capacity $d_G(u)$; each $u \in V_2$ has capacity $\frac{n-k-1}{2}$. This completes the construction. G' is chordal because it is a split graph.

Suppose that G has a dominating set of size k. We delete the same vertices of V_2 and claim that G' becomes orientable. We observe that all vertices of V_1 have at least a deleted neighbor, since we deleted a dominating set of G, hence for each such vertex the number of remaining incident edges is at most its capacity. We therefore orient all edges incident on V_1 towards V_1. Finally, for the remaining vertices of V_2 which induce a clique of size $n - k$ we orient their edges using Lemma 2 so that they all have in-degree exactly $\frac{n-k-1}{2}$.

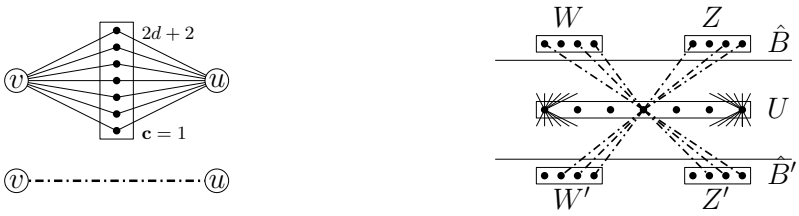


Figure 3 Left: An example OR gadget. In the following, OR gadgets are shown as dotted edges. **Right:** Example connections between a set U and the p sets W, Z in the gadgets $\hat{B}$ of its group.

For the converse direction, suppose we can delete at most k vertices of the new graph to make it orientable respecting the capacities. Again, as in Theorem 9 we assume we have a solution of size exactly k, otherwise we add some vertices. Furthermore, any used vertex of V_1 can be exchanged with one of its neighbors in V_2, since all vertices of V_1 have degree one more than their capacities, hence we assume that the solution deletes k vertices of V_2. We show that these vertices are a dominating set of G. Suppose for contradiction that they are not, so $u \in V_1$ does not have any deleted neighbors in V_2. Since there are $d(u) + 1$ edges connecting $u \in V_1$ to V_2, at least one of them is oriented towards V_2. But now the $n - k$ non-deleted vertices of V_2, because of Lemma 2 all have in-degree exactly equal to the capacities inside the clique they induce. Hence, the additional edge from V_1 will force a vertex to violate its capacity. ◀

▶ **Theorem 11 (★).** *d-Orientable Deletion can be solved in time $d^{O(d+k)}n^{O(1)}$ on chordal graphs, where k is the size of the solution.*

5 SETH Lower Bound for Treewidth

Overview. We follow the approach for proving SETH lower bounds for treewidth algorithms introduced in [16] (see also Chapter 14 in [13]), that is, we present a reduction from SAT to d-Orientable Deletion showing that if there exists a better than $(d+2)^{\text{tw}}$ algorithm for d-Orientable Deletion, we obtain a better than 2^n algorithm for SAT.

Similarly to these proofs, our reduction is based on the construction of "long paths" of *Block gadgets*, that are serially connected in a path-like manner. Each such "path" corresponds to a group of variables of the given formula, while each *column* of this construction is associated with one of its clauses. Intuitively, our aim is to embed the 2^n possible variable assignments into the $(d+2)^{\text{tw}}$ states of some optimal dynamic program that would solve the problem on our constructed instance. The hard part of the reduction is to take the natural $d+2$ options available for each vertex, corresponding to its in-degree $(d+1)$ or the choice to delete it $(+1)$, and use them to compress n boolean variables into roughly $\frac{n}{\log(d+2)}$ units of treewidth.

Below, we present a sequence of gadgets used in our reduction. The aforementioned block gadgets, which allow a solution to choose among $d+2$ reasonable choices, are the main ingredient. We connect these gadgets in a path-like manner that ensures that choices remain consistent throughout the construction, and connect clause gadgets in different "columns" of the constructed grid in a way that allows us to verify if the choice made represents a satisfying assignment, without increasing the graph's treewidth.

OR gadget. We use an OR gadget with two endpoints v, u whose purpose is to ensure that in any optimal solution, either v or u will have to be deleted. This gadget is simply a set of $2d+2$ vertices of capacity 1, connected to both v and u, as shown in Figure 3.

Clause gadget $\hat{C}(N)$. This gadget is identical to the one used for Independent Set in [16] (where all vertices are of capacity 0), as finding a maximum independent set can be seen as equivalent to finding a minimum-sized deletion set for 0-orientability. Due to space restrictions, its construction and proof of correctness are omitted here. The gadget has N *input* vertices and its purpose is to offer an 1-in-N choice, while its pathwidth remains constant. The non-deleted input will correspond to a true literal within the clause.

Block gadget $\hat{B}$. This gadget is the basic building block of our construction:

1. Make three vertices a, a', b. Note that in the final construction, our block gadgets will be connected serially, with vertex a' being identified with the following gadget's vertex a.
2. Make three independent sets $X \coloneqq \{x_1, \ldots, x_d\}$, $Y \coloneqq \{y_1, \ldots, y_d\}$, $Q \coloneqq \{q_1, \ldots, q_{2d+1}\}$.
3. Make two sets $W \coloneqq \{w_0, \ldots, w_{d+1}\}$ and $Z \coloneqq \{z_0, \ldots, z_{d+1}\}$.
4. Connect all vertices of X with vertex a and with all vertices of Q.
5. Connect all vertices of Y with vertex a' and with all vertices of Q.
6. Connect all vertices from W except w_{d+1} to b and all vertices of X.
7. Connect all vertices from Z except z_{d+1} to b and all vertices of Y.
8. Attach OR gadgets between the pairs: a and b, b and a', a and w_{d+1}, a' and z_{d+1}.
9. Attach OR gadgets between any pair of vertices in $W \cup Z$, except for the pairs (w_i, z_i) for $i \in \{0, \ldots, d+1\}$. In other words, $W \cup Z$ is an OR-clique, minus a perfect matching.

We set the capacities as follows (see also Figure 4).

- $\mathbf{c}(a) = \mathbf{c}(a') = d$, $\mathbf{c}(b) = 0$.
- $\forall i \in [1, 2d+1], \mathbf{c}(q_i) = d$, and $\forall i, j \in [1, d], \mathbf{c}(x_i) = \mathbf{c}(y_j) = 0$.
- $\forall i \in [0, d], \mathbf{c}(w_i) = i, \mathbf{c}(z_i) = d - i$, and $\mathbf{c}(w_{d+1}) = \mathbf{c}(z_{d+1}) = 0$.

Intuitively, there are $d+2$ options in each gadget, linked to the circumstances of vertices a, a' and b:[1] there will have to be d vertices deleted in total from $X \cup Y$ and the numbers will be complementary: if i vertices remain in X then, due to Q being of size $2d+1$ (it is never useful to delete any of them), there must be $d - i$ vertices remaining in Y. Thus, the $d+1$ options can be seen as represented by the number of vertices remaining in X, while for each one, vertex b must also be deleted due to the OR gadgets connecting it to a, a'. The extra option is to ignore the actual number of deletions within X and remove both a, a' instead.

The sets W, Z are connected in such a way that any reasonable feasible solution will delete all of their vertices, except for a pair w_i, z_i for some $i \in \{0, \ldots, d+1\}$. The non-deleted pair is meant to encode a choice for this block gadget.

Global construction. Fix some integer d, and suppose that for some $\epsilon > 0$ there exists a $(d+2-\epsilon)^{\mathrm{tw}}$ algorithm for d-Orientable Deletion. We give a reduction which, starting from any SAT instance with n variables and m clauses, produces an instance of d-Orientable Deletion, such that applying this supposed algorithm on the new instance would give a better than 2^n algorithm for SAT.

We are faced with the problem that $d+2$ is not a power of 2, hence we will need to create a correspondence between groups of variables of the SAT instance and groups of block gadgets. We first choose an integer $p = \lceil \frac{1}{(1-\lambda)\log_2(d+2)} \rceil$, for $\lambda = \log_{d+2}(d+2-\epsilon) < 1$. We

[1] Each such option can be seen to correspond with one of the states that some optimal dynamic programming algorithm for the problem would assign to vertex a: it is either deleted, or has a number $i \in [0, d]$ of incoming edges within the gadget.

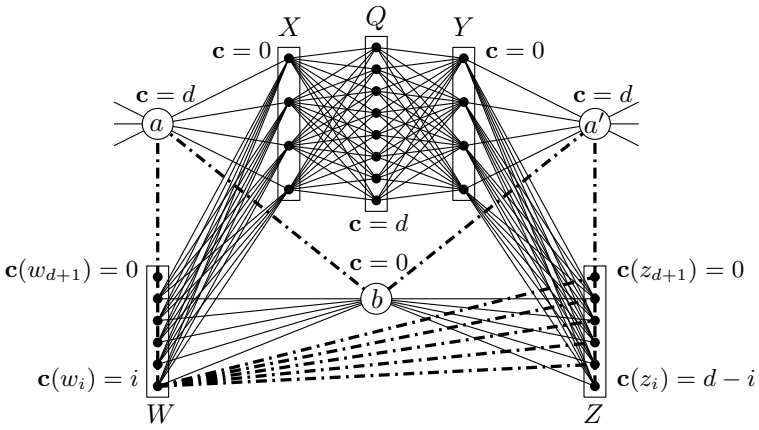


Figure 4 Our block gadget $\hat{B}$. Capacities are shown next to vertices/sets, the OR-connections within W, Z are shown as paths, while the OR-connections between W, Z are only shown for w_0.

then group the variables of ϕ into $t = \lceil \frac{n}{\gamma} \rceil$ groups $F_1, \ldots, F_t$, where $\gamma = \lfloor \log_2(d+2)^p \rfloor$ is the maximum size of each group. Our construction then proceeds as follows (see Figure 5):

1. Make a group of p block gadgets $\hat{B}_\tau^{1,\pi}$ for $\pi \in [1,p]$, for each group F_τ of variables of ϕ with $\tau \in [1,t]$.
2. Make a clique $U_\tau^1 := \{u_\tau^{1,1}, \ldots, u_\tau^{1,(d+2)^p}\}$ on $(d+2)^p$ vertices, whose capacities are all set to 0, for each group F_τ of variables of ϕ with $\tau \in [1,t]$.
3. Associate each of these $(d+2)^p$ vertices from U_τ^1 with one of the $d+2$ options *for each* gadget $\hat{B}_\tau^{1,\pi}$, i.e. over all $\pi \in [1,p]$.
4. Connect each $u_\tau^{1,i}$ for $i \in [1,(d+2)^p]$ to each vertex from each W and Z within each of the p gadgets $\hat{B}$ that *do not match* the option associated with $u_\tau^{1,i}$ via OR gadgets (see Figure 3 for an example).
5. Make $m(tpd+2)$ copies of this first "column" of gadgets.
6. Identify each vertex a' in $\hat{B}_\tau^{l,\pi}$ with the vertex a of its following gadget $\hat{B}_\tau^{l+1,\pi}$, i.e. for fixed $\tau \in [1,t]$ and $\pi \in [1,p]$, all block gadgets are connected in a path-like manner.
7. For every clause C_μ, with $\mu \in [1,m]$, make a clause gadget $\hat{C}_\mu^o$ with $N = q_\mu$ inputs, where q_μ is the number of literals[2] in clause C_μ and $o \in [0, tpd+1]$.
8. For every $\tau \in [1,t]$, associate one of the $(d+2)^p$ vertices of U_τ^l (that is in turn associated with one of $d+2$ options for each of the p gadgets of group F_τ), with an assignment to the variables in group F_τ. Note that as there are at most $2^\gamma = 2^{\lfloor \log_2(d+2)^p \rfloor}$ assignments to the variables in F_τ and $(d+2)^p \geqslant 2^\gamma$ such vertices, the association can be unique for each τ (and the same for all $l \in [1, m(tpd+2)]$).
9. Each of the clause gadget's q_μ inputs will correspond to a literal appearing in clause C_μ.
10. Connect via OR gadgets each input from each $\hat{C}_\mu^o$, corresponding to a literal whose variable appears in group F_τ, to the all vertices from the set $U_\tau^{mo+\mu}$ (in its appropriate column) whose associated assignments *do not satisfy* the input's literal.

▶ **Theorem 12 (★).** *For any fixed $d \geqslant 1$, if d-*Orientable Deletion *can be solved in $O^*((d+2-\epsilon)^{tw(G)})$ time for some $\epsilon > 0$, then there exists some $\delta > 0$, such that* SAT *can be solved in $O^*((2-\delta)^n)$ time.*

[2] We assume that q_μ is always even, by duplicating some literals if necessary.

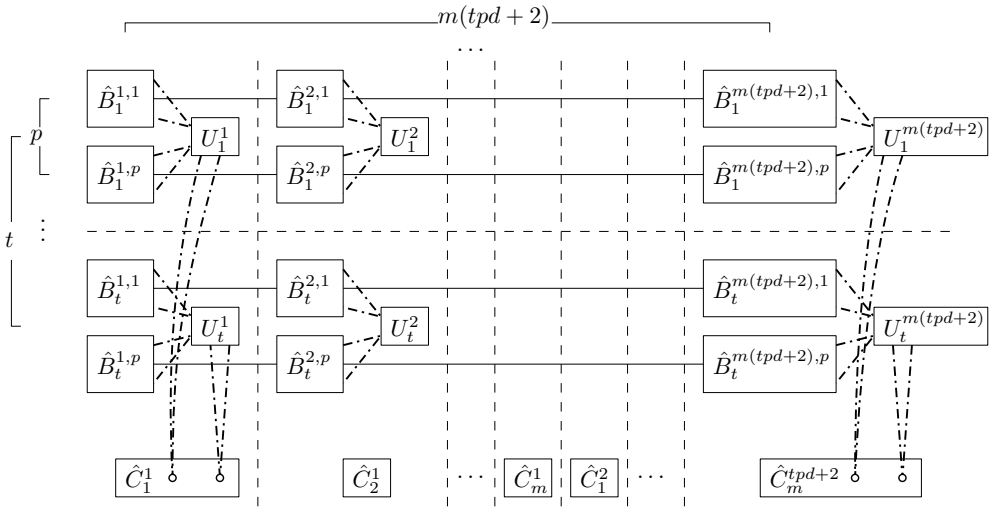


Figure 5 A simplified picture of the complete construction.

▶ **Corollary 13.** *If* Pseudoforest Deletion *can be solved in* $O^*((3-\epsilon)^{tw(G)})$ *time for some* $\epsilon > 0$*, then there exists some* $\delta > 0$*, such that* SAT *can be solved in* $O^*((2-\delta)^n)$ *time.*

6 Algorithm for Clique-Width

In this section we present a dynamic programming algorithm for d-Orientable Deletion parameterized by the clique-width of the input graph, of running time $d^{O(d \cdot \mathrm{cw})}$. The algorithm is based on the dynamic programming of [9] for Max W-Light, the problem of assigning a direction to each edge of an undirected graph so that the number of vertices of out-degree at most W is maximized. As noted in [9] (and our Section 1), that problem is supplementary to Min $(W+1)$-Heavy, the problem of minimizing the number of vertices of out-degree at least $W+1$ in terms of exact computation (though their approximability properties may vary), that in turn can be seen as the optimization version of d-Orientable Deletion for $d = W$, if we simply consider the in-degree of every vertex instead of the out-degree (by reversing the direction of every edge in any given orientation).

The dynamic programming algorithm of [9] runs in XP-time $n^{O(d \cdot \mathrm{cw})}$, by considering the full number of possible states for each label of a clique-width expression T for the input graph G:[3] for each node t of T, it computes an *in-degree-signature* of G_t, being a table $A_t = (A_t^{i,j}), \forall i \in [1, \mathrm{cw}], j \in [0, d]$, if there is an orientation Λ_t (of every edge of G_t) such that for each label $i \in [1, \mathrm{cw}]$ and *in-degree-class* $j \in [0, d]$, the entry $A_t^{i,j}$ is the number of vertices labelled i with in-degree j in G_t under Λ_t, and also a *deletion set* K_t, where $K_t := \bigcup_{i \in [1,\mathrm{cw}]} K_t^i$ for each $i \in [1, \mathrm{cw}]$, where K_t^i is the set of vertices labelled i that are deleted from G_t. Based on this scheme, the updating process of the tables is straightforward for Leaf, Relabel and Union nodes, while for Join nodes, the computation of the degree signatures is based on a result by [1], stating that an orientation satisfying any given lower and upper in-degree bounds for each vertex can be computed in $O(m^{3/2} \log n)$ time, where m is the number of edges to be oriented. We refer to [9] for details.

[3] Slightly paraphrased here for d-Orientable Deletion, keeping the same notation.

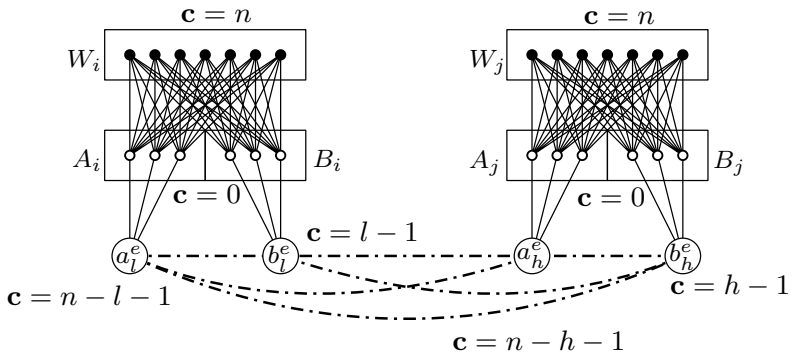


Figure 6 A partial view of the construction, depicting the gadgets encoding the selection for V_i, V_j, as well the representation of an edge $e = (v_i^l, v_j^h)$. Note dotted edges signifying OR gadgets.

The aim of this section is to improve the running time of the above algorithm to $d^{O(d \cdot \mathrm{cw})}$, that is FPT-time parameterized by d and cw, by showing that not all of the natural states utilized therein are in fact required. The main idea behind this improvement is based on the redundancy of *exactly* keeping track of the size of an in-degree-class above a certain threshold (i.e. d^4), since the valid d-orientations of a biclique created after joining such an in-degree-class with some other label are greatly constrained, as any optimal solution will always orient all new edges towards the vertices of this "large" class in order to maintain d-orientability (and update in-degree-class sizes accordingly), while respecting the given deletion set and orientation of previously introduced edges.

▶ **Theorem 14 (★).** *Given a graph G along with a cw-expression T of G, the d-*Orientable Deletion *problem can be solved in time $O^*(d^{O(d \cdot cw)})$.*

7 W-hardness for Clique-Width

In this section we present a reduction establishing that the algorithm of Section 6 is essentially optimal. More precisely, we show that, under the ETH, no algorithm can solve d-Orientable Deletion in time $n^{o(\mathrm{cw})}$. As a result, the parameter dependence of $d^{O(\mathrm{cw})}$ of the algorithm in Section 6 cannot be improved to a function that only depends on cw. We prove this result through a reduction from k-Multicolored Independent Set. As before, we employ capacities and implicitly utilize Capacitated-d-Orientable Deletion.

Construction. Recall that an instance $[G = (V, E), k]$ of k-Multicolored Independent Set consists of a graph G whose vertex set is given to us partitioned into k sets $V_1, \ldots V_k$, with $|V_i| = n$ for all $i \in [1, k]$, and with each V_i inducing a clique. Given such an instance, we will construct an instance $G' = (V', E')$ of d-Orientable Deletion, where $d = n$. Let $V_i \coloneqq \{v_i^1, \ldots, v_i^n\}, \forall i \in [1, k]$. To simplify notation, we use E to denote the set of *non-clique* edges, i.e. those connecting vertices in parts V_i, V_j for $i \neq j$. Our construction is given as follows, while Figure 6 provides an illustration:

1. Create two sets $A_i, B_i \subset V', \forall i \in [1, k]$ of n vertices each, of capacities 0.
2. Make a set of *guard* vertices $W_i, \forall i \in [1, k]$, of size $kn + 3|E| + 1$, of capacities n.
3. Connect each vertex of W_i to all vertices of A_i, B_i for all $i \in [1, k]$.

4. For each edge $e = (v_i^l, v_j^h) \in E$ with endpoints $v_i^l \in V_i, v_j^h \in V_j$ (i.e. the l-th vertex of V_i and the h-th vertex of V_j), make four new vertices $a_l^e, b_l^e, a_h^e, b_h^e$.
5. Connect $a_l^e, b_l^e, a_h^e, b_h^e$ to each other via OR gadgets.
6. Connect a_l^e to all vertices of A_i and b_l^e to all vertices of B_i, while a_h^e is connected to all vertices of A_j and b_h^e to all vertices of B_j.
7. Set the capacities $\mathbf{c}(a_l^e) = n - l - 1$, $\mathbf{c}(b_l^e) = l - 1$, $\mathbf{c}(a_h^e) = n - h - 1$ and $\mathbf{c}(b_h^e) = h - 1$.

▶ **Theorem 15 (★).** *d-Orientable Deletion is W[1]-hard parameterized by the clique-width of the input graph. Furthermore, if there exists an algorithm solving d-Orientable Deletion in time $n^{o(cw)}$ then the ETH is false.*

References

1 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Upper and lower degree bounded graph orientation with minimum penalty. In *CATS'12*, volume 128, pages 139–145, 2012.

2 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, and Hirotaka Ono. Degree-constrained graph orientation: Maximum satisfaction and minimum violation. *Theory Comput. Syst.*, 58(1):60–93, 2016. doi:10.1007/s00224-014-9565-5.

3 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *J. Comb. Optim.*, 22(1):78–96, 2011.

4 Yuichi Asahiro, Eiji Miyano, and Hirotaka Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. *D.A.M.*, 159(7):498–508, 2011.

5 Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *Int. J. Found. Comput. Sci.*, 18(2):197–215, 2007. doi:10.1142/S0129054107004644.

6 MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC*, pages 543–552. ACM, 2009.

7 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012.

8 Hans L. Bodlaender, Hirotaka Ono, and Yota Otachi. A faster parameterized algorithm for pseudoforest deletion. In *IPEC*, volume 63, pages 7:1–7:12, 2016.

9 Hans L. Bodlaender, Hirotaka Ono, and Yota Otachi. Degree-constrained orientation of maximum satisfaction: Graph classes and parameterized complexity. *Algorithmica*, 80(7):2160–2180, 2018. doi:10.1007/s00453-017-0399-9.

10 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *FOCS*, pages 107–116. IEEE Computer Society, 2009.

11 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86(2):243–266, 1991.

12 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.

15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

16 D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789, 2011.

17 Luke Mathieson. The parameterized complexity of editing graphs for bounded degeneracy. *Theor. Comput. Sci.*, 411(34-36):3181–3187, 2010.

18 Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. In *MFCS (2)*, volume 9235 of *LNCS*, pages 517–528, 2015.

19 Stefan Szeider. Not so easy problems for tree decomposable graphs. *CoRR,abs/1107.1177*, 2011.

20 José Verschae and Andreas Wiese. On the configuration-lp for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014. doi:10.1007/s10951-013-0359-4.

SVM via Saddle Point Optimization: New Bounds and Distributed Algorithms

Lingxiao Huang
École polytechnique fédérale de Lausanne
CH-1015, Lausanne, Switzerland
lingxiao.huang@epfl.ch

Yifei Jin
Tsinghua University
Beijing 100084, China
jin-yf13@mails.tsinghua.edu.cn

Jian Li
Tsinghua University
Beijing 100084, China
Corresponding author: lijian83@mail.tsinghua.edu.cn

Abstract

We study two important SVM variants: hard-margin SVM (for linearly separable cases) and ν-SVM (for linearly non-separable cases). We propose new algorithms from the perspective of saddle point optimization. Our algorithms achieve $(1-\epsilon)$-approximations with running time $\tilde{O}(nd + n\sqrt{d/\epsilon})$ for both variants, where n is the number of points and d is the dimensionality. To the best of our knowledge, the current best algorithm for ν-SVM is based on quadratic programming approach which requires $\Omega(n^2 d)$ time in worst case [Joachims, 1998; Platt, 1999]. In the paper, we provide the first nearly linear time algorithm for ν-SVM. The current best algorithm for hard margin SVM achieved by Gilbert algorithm [Gärtner and Jaggi, 2009] requires $O(nd/\epsilon)$ time. Our algorithm improves the running time by a factor of $\sqrt{d}/\sqrt{\epsilon}$. Moreover, our algorithms can be implemented in the distributed settings naturally. We prove that our algorithms require $\tilde{O}(k(d+\sqrt{d/\epsilon}))$ communication cost, where k is the number of clients, which almost matches the theoretical lower bound. Numerical experiments support our theory and show that our algorithms converge faster on high dimensional, large and dense data sets, as compared to previous methods.

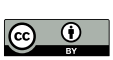
2012 ACM Subject Classification Mathematics of computing → Continuous optimization, Computing methodologies → Support vector machines

Keywords and phrases ν-SVM, hard-margin SVM, saddle point optimization, distributed algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.25

Related Version A full version of the paper is available at `https://arxiv.org/abs/1705.07252`.

Acknowledgements This research is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia.



1 Introduction

Support Vector Machine (SVM) is widely used for classification in numerous applications such as text categorization, image classification, and hand-written characters recognition.

In this paper, we focus on binary classification. If two classes of points which are linearly separable, one can use the hard-margin SVM [7, 9], which is to find a hyperplane that separate two classes of points and the margin is maximized. If the data is not linearly separable, several popular SVM variants have been proposed, such as l_2-SVM, C-SVM and ν-SVM (see e.g., the summary in [15]). The main difference among these variants is that they use different penalty loss functions for the misclassified points. l_2-SVM, as the name implied, uses the l_2 penalty loss. C-SVM and ν-SVM are two well-known SVM variants using l_1-loss. C-SVM uses the l_1-loss with penalty coefficient $C \in [0, \infty)$ [43]. On the other hand, ν-SVM reformulates C-SVM through taking a new regularization parameter $\nu \in (0, 1]$ [35]. However, given a C-SVM formulation, it is not easy to compute the regularization parameter ν and obtain an equivalent ν-SVM. Because the equivalence is based on some hard-to-compute constant. Compared to C-SVM, the parameter ν in ν-SVM has a more clear geometric interpretation: the objective is to minimize the distance between two reduced polytopes defined based on ν [10]. However, the best known algorithm for ν-SVM is much worse than that for C-SVM in practice (see below).

In general, SVMs can be formulated as convex quadratic programs and solved by quadratic programs in $O(n^2 d)$ time [21, 34]. However, better algorithms exists for some SVM variants, which we briefly discuss below.

For hard-margin SVM, [15] showed that Gilbert algorithm [16] achieves a $(1 - \epsilon)$-approximation with $O(nd/\epsilon\beta^2)$ running time where β is the ratio of the minimum distance to the maximum one among the points. l_2-SVM and C-SVM have been studied extensively and current best algorithms runs in time linear in the number n of data points [2, 6, 11, 13, 36]. Moreover, if the parameter C is sufficiently small, e.g., $C = \Theta(1/n)$, then C-SVM can be solved in $\tilde{O}(d/\epsilon)$ time [36], which is independent of n. However, these techniques cannot be extended to ν-SVM directly, mainly because ν-SVM cannot be transformed to single-objective unconstrained optimization problems. Except the traditional quadratic programming approach, there is no better algorithm known with provable guarantee for ν-SVM. Whether ν-SVM can be solved in nearly linear time is still open.

Distributed SVM has also attracted significant attention in recent years. A number of distributed algorithms for SVM have been obtained in the past [12, 17, 28, 30, 41]. Typically, the communication complexity is one of the key performance measurements for distributed algorithms, and has been studied extensively (see [25, 32, 40]). For hard-margin SVM, Liu et al. [26] proposed a distributed algorithm with $O(kd/\epsilon)$ communication cost, where k is the number of the clients. Hence, it is a natural question to ask whether the communication cost of their algorithm can be improved.

1.1 Our Contributions

We summarize our main contributions as follows.

1. **Hard-Margin SVM:** We provide a new $(1 - \epsilon)$-approximation algorithm with running time $\tilde{O}(nd+n\sqrt{d}/\sqrt{\epsilon\beta})$, where β is the ratio of the minimum distance to the maximum one among the points (see Theorem 9).[1] Compared to Gilbert algorithm [15], our algorithm

[1] $\tilde{O}$ notation hides logarithm factors such as $\log(n)$, $\log(\beta)$ and $\log(1/\epsilon)$.

improves the running time by a factor of $\sqrt{d}/\sqrt{\epsilon}$. First, we regard hard-margin SVM as computing the polytope distance between two classes of points. Then we translate the problem to a saddle point optimization problem using the properties of the geometric structures (Lemma 2), and provide an algorithm to solve the saddle point optimization.

2. **ν-SVM:** Then, we extend our algorithm to ν-SVM and design an $\tilde{O}(nd+n\sqrt{d}/\sqrt{\epsilon\beta})$ time algorithm, which is the most important technical contribution of this paper. To the best of our knowledge, it is the first nearly linear time algorithm for ν-SVM. It is known that ν-SVM is equivalent to computing the distance between two reduced polytopes [10, 5]. The obstacle for providing an efficient algorithm based on the reduced polytopes is that the number of vertices in the reduced polytopes may be exponentially large. However, in our framework, we only need to implicitly represent the reduced polytopes. We show that using the similar saddle point optimization framework, together with a new nontrivial projection method, ν-SVM can be solved efficiently in the same time complexity as in the hard-margin case. Compared with the QP-based algorithms in previous work [21, 34], our algorithm significantly improves the running time, by a factor of n.
3. **Distributed SVM:** Finally, we extend our algorithms for both hard-margin SVM and ν-SVM to the distributed setting. We prove that the communication cost of our algorithm is $\tilde{O}(k(d+\sqrt{d/\epsilon}))$, which is almost optimal according to the lower bound provided in [26]. For the hard-margin SVM, compared with the current best algorithm [26] with $O(kd/\epsilon)$ communication cost, our algorithm is more suitable when ϵ is small and d is large. For ν-SVM, our algorithm is the first practical distributed algorithm.

Besides, the numerical experiments support our theoretical bounds. We compare our algorithms with Gilbert Algorithm [15] and NuSVC in scikit-learn [33]. The experiments show that our algorithms converge faster on high dimensional, large and dense data sets. See the full version for the details.

1.2 Other Related Work

For the hard-margin SVM, there is an alternative to Gilbert's method, called the MDM algorithm, originally proposed by [29]. Recently, López and Dorronsoro proved that the rate of convergence of MDM algorithm is $O(n^2 d\log(1/\epsilon))$ [27] which is a linear convergence w.r.t. ϵ, but worse than Gilbert Algorithm w.r.t. n.

Both C-SVM and l_2-SVM have been studied extensively in the literature. Basically, there are three main algorithmic approaches: the primal gradient-based methods [24, 36, 13, 11, 2], dual quadratic programming methods [22, 37, 20] and dual geometry methods [39, 38]. Recently, [2] provided the current best algorithms which achieve $O(nd/\sqrt{\epsilon})$ time for l_2-SVM and $O(nd/\epsilon)$ time for C-SVM.

Sublinear time algorithms for hard-margin SVM and l_2-SVM have been proposed [8, 19]. These algorithms are sublinear w.r.t. nd, (i.e., the size of the input), but have worse dependency on $1/\epsilon$.

The algorithmic framework for saddle point optimization was first developed by Nesterov for structured nonsmooth optimization problem [31]. He only considered the full gradient in the algorithm. Recently, some studies have extended it to the stochastic gradient setting [42, 3]. The most related work is [3], in which the author obtained an $\tilde{O}(nd+n\sqrt{d}/\sqrt{\epsilon})$ algorithm for the minimum enclosing ball problem (MinEB) in Euclidean space, using the saddle point optimization. This result also implies an algorithm for l_2-SVM, by the connection between MinEB and l_2-SVM (see [39, 38, 18]). However, the implied algorithm is not as efficient. Based on [39, 38], the dual of l_2-SVM is equivalent to MinEB by a specific feature mapping.

It maps a d-dimensional point to the $(d+n)$-dimensional space. Thus, after the mapping, it takes quadratic time to solve l_2-SVM. To avoid this mapping, they designed an algorithm called Core Vector Machine (CVM), in which they can solve l_2-SVM by solving $O(1/\epsilon)$ MinEB problems sequentially.

2 Formulate SVM as Saddle Point Optimization

In this section, we formulate both hard-margin SVM and ν-SVM, and show that they can be reduced to saddle point optimizations. All vectors in the paper are all column vectors by default.

▶ **Definition 1** (Hard-margin SVM)**.** Given n points $x_i \in \mathbb{R}^d$ for $1 \le i \le n$, each x_i has a label $y_i \in \{\pm 1\}$. The hard-margin SVM can be formalized as the following quadratic programming [9].

$$\begin{aligned} \min_{w,b} \quad & \tfrac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^{\mathrm{T}}x_i - b) \ge 1, \quad \forall i \end{aligned} \tag{1}$$

The dual problem of (1) is defined as follows, which is equivalent to finding the minimum distance between the two convex hulls of two classes of points when they are linearly separable [5]. We call the problem the C-Hull problem.

$$\begin{aligned} \min_{\eta,\xi} \quad & \tfrac{1}{2}\|A\eta - B\xi\|^2 \\ \text{s.t.} \quad & \|\eta\|_1 = 1, \|\xi\|_1 = 1. \quad \eta \ge 0, \xi \ge 0. \end{aligned} \tag{2}$$

where A and B are the matrices in which each column represents a vector of a point with label $+1$ or -1 respectively.

Denote the set of points with label $+1$ by $\mathcal{P}$ and the set with label -1 by $\mathcal{Q}$. Let $n_1 = |\mathcal{P}|$ and $n_2 = |\mathcal{Q}|$. Since $\sum_i \eta_i = 1$, we can regard it as a probability distribution among points in P (similarly for Q). We denote Δ_{n_1} to be the set of n_1-dimensional probability vectors over $\mathcal{P}$ and Δ_{n_2} to be that over $\mathcal{Q}$. Then, we prove that the C-Hull problem (2) is equivalent to the following saddle point optimization in Lemma 2.

▶ **Lemma 2.** *Problem C-Hull* (2) *is equivalent to the saddle point optimization* (3)*.*

$$\mathsf{OPT} = \max_{w} \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2 \tag{3}$$

Proof. Consider the saddle point optimization (3). First, note that

$$w^{\mathrm{T}}A\eta - w^{\mathrm{T}}B\xi - \frac{1}{2}\|w\|^2 = w^{\mathrm{T}}(A\eta - B\xi) - \frac{1}{2}\|w\|^2$$

The range of the term $(A\eta - B\xi)$ for $\eta \in \Delta_{n_1}, \xi \in \Delta_{n_2}$ is a convex set, denoted by $\mathcal{S}$. Since the convex hulls of $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable, we have $0 \notin \mathcal{S}$. Denote $\phi(w,z) = w^{\mathrm{T}}z - \frac{1}{2}\|w\|^2$ for any $w \in \mathbb{R}^d, z \in \mathcal{S}$. Then (3) is equivalent to $\max_w \min_{z\in\mathcal{S}} \phi(w,z)$. Note that

$$\max_{w} \min_{z\in\mathcal{S}} \phi(w,z) \ge \min_{z\in\mathcal{S}} \phi(\mathbf{0}^d, z) = 0.$$

Thus, we only need to consider those directions $w \in \mathbb{R}^d$ such that there exists a point $z \in \mathcal{S}$ with $w^T z \ge 0$. We use $\mathcal{W}$ to denote the collection of such directions.

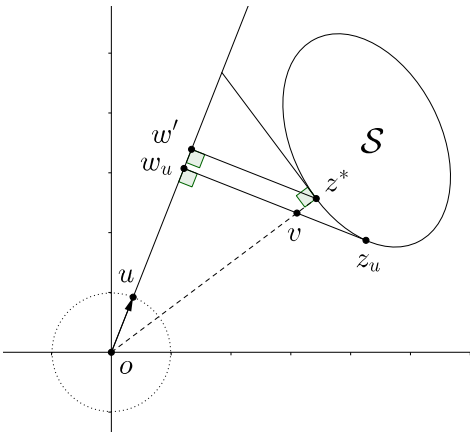


Figure 1 The equivalence between C-Hull and saddle point optimization (3).

Let u be a unit vector in $\mathcal{W}$. Denote

$$z_u := \arg\min_{z\in\mathcal{S}} \phi(u, z) = \arg\min_{z\in\mathcal{S}} u^T z.$$

By this definition, z_u is the point with smallest projection distance to u among $\mathcal{S}$ (see Figure 1). Observe that if a direction $w = c \cdot u$ $(c > 0)$, then we have $\arg\min_z \phi(w, z) = \arg\min_z \phi(u, z)$. Also note that

$$\max_{w=c\cdot u:c>0} w^{\mathrm{T}} z_u - \frac{1}{2}\|w\|^2 = \max_{w=c\cdot u:c>0} \frac{1}{2}(-\|w - z_u\|^2 + \|z_u\|^2).$$

Let $w_u := \arg\max_{w=c\cdot u:c>0} \phi(w, z_u) = \arg\min_{w=c\cdot u:c>0} \|w - z_u\|^2$ be the projection point of z_u to the line ou, where o is the origin. See Figure 1 for an example. Overall, we have

$$\begin{aligned}\max_{w} \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} w^{\mathrm{T}}(A\eta - B\xi) - \frac{1}{2}\|w\|^2 &= \max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}(-\|w_u - z_u\|^2 + \|z_u\|^2) \\ &= \max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}\|w_u\|^2.\end{aligned}$$

The last equality is by the Pythagorean theorem. Let z^* be the closest point in $\mathcal{S}$ to the origin point. Next, we show that $\max_{u\in\mathcal{W}:\|u\|=1} \|w_u\|^2 = \|z^*\|^2$. Given a unit vector $u \in \mathcal{W}$, define w' to be the projection point of z^* to the line ou. By the definition of z_u and w_u, we have that $\max_u \|w_u\|^2 \le \|w'\|^2 \le \|z^*\|^2$. Moreover, let $u = z^*/\|z^*\|$. In this case, we have $\|w_u\|^2 = \|z^*\|^2$. Thus, we conclude that $\max_u \|w_u\|^2 = \|z^*\|^2$.

Overall, we prove that

$$\max_{u\in\mathcal{W}:\|u\|=1} \frac{1}{2}\|w_u\|^2 = \frac{1}{2}\|z^*\|^2 \min_{z\in\mathcal{S}} \frac{1}{2}\|z\|^2 = \min_{\eta\in\Delta_{n_1},\xi\in\Delta_{n_2}} \frac{1}{2}\|A\eta - B\xi\|^2.$$

Thus, C-Hull (2) is equivalent to the saddle point optimization (3). ◀

Let $\phi(w, \eta, \xi) = w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi - \|w\|^2/2$. Note that $\phi(w, \eta, \xi)$ is only linear w.r.t. η and ξ. However, in order to obtain an algorithm which converges faster, we hope that the objective function is strongly convex with respect to η and ξ. For this purpose, we can add a small

regularization term which ensures that the objective function is strongly convex. This is a commonly used approach in optimization (see [3] for an example). Here, we use the entropy function $H(u) := \sum_i u_i \log u_i$ as the regularization term. The new saddle point optimization problem is as follows.

$$\max_{w} \min_{\eta \in \Delta_{n_1}, \xi \in \Delta_{n_2}} \quad w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi + \gamma H(\eta) + \gamma H(\xi) - \tfrac{1}{2}\|w\|^2, \tag{4}$$

where $\gamma = \epsilon\beta/2\log n$. The following lemma describes the efficiency of the above saddle point optimization (4). We defer the proof to the full version.

▶ **Lemma 3.** *Let (w^*, η^*, ξ^*) and $(w^\circ, \eta^\circ, \xi^\circ)$ be the optimal solution of saddle point optimizations* (3) *and* (4) *respectively. Define* OPT *as in* (3)*. Define*

$$g(w) := \min_{\eta \in \Delta_{n_1}, \xi \in \Delta_{n_2}} w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi - \frac{1}{2}\|w\|^2.$$

Then $g(w^*) - g(w^\circ) \le \epsilon\mathsf{OPT}$ (note that $g(w^*) = \mathsf{OPT}$).

We call the saddle point optimization (4) the Hard-Margin Saddle problem, abbreviated as HM-Saddle. Next, we discuss ν-SVM (see [35, 10]) and again provide an equivalent saddle point optimization formulation.

▶ **Definition 4** (ν-SVM)**.** Given n points $x_i \in \mathbb{R}^d$ for $1 \le i \le n$, each x_i has a label $y_i \in \{+1, -1\}$. ν-SVM is the quadratic programming as follows.

$$\begin{array}{ll} \min\limits_{w,b,\rho,\delta} & \frac{1}{2}\|w\|^2 - \rho + \frac{\nu}{2}\sum_i \delta_i \\ \text{s.t.} & y_i(w^{\mathrm{T}} x_i - b) \ge \rho - \delta_i, \delta_i \ge 0, \quad \forall i \end{array} \tag{5}$$

Crisp and Burges [10] presented a geometry interpretation for ν-SVM. They proved ν-SVM is equivalent to the following problem of finding the closest distance between two reduced convex hulls.

$$\begin{array}{ll} \min\limits_{\eta,\xi} & \frac{1}{2}\|A\eta - B\xi\|^2 \\ \text{s.t.} & \|\eta\|_1 = 1,\ \|\xi\|_1 = 1,\ 0 \le \eta_i \le \nu,\ 0 \le \xi_j \le \nu,\ \forall i, j \end{array} \tag{6}$$

We call the above problem the Reduced Convex Hull problem, abbreviated as RC-Hull. The difference between C-Hull (2) and RC-Hull (6) is that in the latter one, each entry of η and ξ has an upper bound ν. Geometrically, it means to compress the convex hull of $\mathcal{P}$ and $\mathcal{Q}$ such that the two reduced convex hulls are linearly separable. We define $\mathcal{D}_{n_1}$ to be the domain of η in RC-Hull, i.e., $\{\eta \mid \|\eta\|_1 = 1, 0 \le \eta_i \le \nu, \forall i\}$ and $\mathcal{D}_{n_2}$ to be the domain of ξ, i.e., $\{\xi \mid \|\xi\|_1 = 1, 0 \le \xi_j \le \nu, \forall j\}$. Similar to Lemma 2, we have the following lemma.

▶ **Lemma 5.** *RC-Hull* (6) *is equivalent to the following saddle point optimization.*

$$\mathsf{OPT} = \max_{w} \min_{\eta \in \mathcal{D}_{n_1},\ \xi \in \mathcal{D}_{n_2}} w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi - \frac{1}{2}\|w\|^2. \tag{7}$$

Proof. The proof is almost the same to the proof of Lemma 2. The only difference is that the range of the term $(A\eta - B\xi)$ is another convex set defined by $\eta \in \mathcal{D}_{n_1}, \xi \in \mathcal{D}_{n_2}$. ◀

Again, we add two entropy terms to make the objective function strongly convex with respective to η and ξ.

$$\max_{w} \min_{\eta \in \mathcal{D}_{n_1}, \xi \in \mathcal{D}_{n_2}} \quad w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi + \gamma H(\eta) + \gamma H(\xi) - \tfrac{1}{2}\|w\|^2. \tag{8}$$

where $\gamma = \epsilon\beta/(2\log n)$. We call this problem a ν-Saddle problem. Similar to Lemma 3, we have the following lemma which states that ν-Saddle (8) is a $(1-\epsilon)$-approximation of the saddle point optimization (7). The proof can be found in the full version.

▶ **Lemma 6.** *Let* (w^*, η^*, ξ^*) *and* $(w^\circ, \eta^\circ, \xi^\circ)$ *be the optimal solution of saddle point optimizations* (7) *and* (8) *respectively. Define* OPT *as in* (7). *Define*

$$g(w) := \min_{\eta\in\mathcal{D}_{n_1}, \xi\in\mathcal{D}_{n_2}} w^{\mathrm{T}} A\eta - w^{\mathrm{T}} B\xi - \frac{1}{2}\|w\|^2.$$

Then $g(w^*) - g(w^\circ) \leq \epsilon\mathsf{OPT}$.

Overall, we formulate hard-margin SVM and ν-SVM as saddle point problems and prove that through solving HM-Saddle and ν-Saddle, we can solve hard-margin SVM and ν-SVM.[2]

3 Saddle Point Optimization Algorithms for SVM

In this section, we propose efficient algorithms to solve the two saddle point optimizations: HM-Saddle (4) and ν-Saddle (8). The framework is inspired by the prior work by [3]. However, their algorithm does not imply an effective SVM algorithm directly as discussed in Section 1.2. We modify the *update rules* and introduce new *projection methods* to adjust the framework to the HM-Saddle and ν-Saddle problems. We highlight that both the new update rules and projection methods are non-trivial.

First, we introduce a preprocess step to make the data vectors more homogeneous in each coordinate. Then, we explain the update rules and projection methods of our algorithm: Saddle-SVC.

For convenience, we assume that in the hard margin case $\|x_i\|^2 \leq 1$ for $1 \leq i \leq n$. [3] Let W be the $d \times d$ Walsh-Hadamard matrix and D be a $d \times d$ diagonal matrix whose entries are i.i.d. chosen from ± 1 with equal probability. We transform the data by left-producting the matrix WD. Then with high probability, for any point x_i satisfied that [1]

$$\forall j \in [d],\ |(WDx_i)_j| \leq O(\sqrt{\log n/d}).$$

Let $X^+ = WDA$ and $X^- = WDB$. It means that after transformation, with high probability, the value of each entry in X^+ or X^- is at most $O(\sqrt{\log n/d})$. This transformation can be completed in $O(nd\log d)$ time by FFT. Note that WD is an invertible matrix which represents a rotation and mirroring operation. Hence, it does not affect the optima of the problem. In fact, the "Hadamard transform trick" has been used in the numerical analysis literature explicitly or implicitly (see e.g., [14, 23, 3]). Roughly speaking, the main purpose of the transform is to make all coordinates of X more uniform, such that the uniform sampling (line 1 in Algorithm 2) is more efficient (otherwise, the large coordinates would have a disproportionate effect on uniform sampling).

After the data transformation, we define some necessary parameters. See line 4 of Algorithm 1 for details.[4] We use "$\alpha[t]$" to represent the value of variable "α" at iteration t.

---

[2] Some readers may wonder why the formulations of HM-Saddle and ν-Saddle only depends on (w, η, ξ) but not the offset b. In fact, according to the fact that the hyperplane bisects the closest points in the (reduced) convex hulls, it is not difficult to show that $b^* = w^{*\mathrm{T}}(A\eta^* + B\xi^*)/2$.

[3] It can be achieved by scaling all data by factor $1/\max\|x_i\|^2$ in $O(nd)$ time.

[4] Careful readers may notice that $\gamma = \epsilon\beta/(2\log n)$. But β is an unknown parameter, which is the ratio of the minimum distance to the maximum one among the points. The same issue also appears in the previous work [3]. The role of β is similar to the step size in the stochastic gradient descent algorithm. In practice, we could try several $\beta = 10^{-k}$ for $k \in \mathbb{Z}$ and choose the best one.

Algorithm 1 Pre-processing

Input: $\mathcal{P}$: n_1 points x_i^+ with label $+1$ and $\mathcal{Q}$: n_2 points x_i^- with label -1

1: $W \leftarrow d$-dimensional Walsh-Hadamard Matrix
2: $D \leftarrow d \times d$ diagonal matrix whose entries are i.i.d. chosen from ± 1
3: $X^+ \leftarrow WD \cdot [x_1^+, x_2^+, \ldots, x_{n_1}^+]$, $X^- \leftarrow WD \cdot [x_1^-, x_2^-, \ldots, x_{n_2}^-]$
4: $\gamma \leftarrow \frac{\epsilon\beta}{2\log n}$, $q \leftarrow O(\sqrt{\log n})$, $\tau \leftarrow \frac{1}{2q}\sqrt{\frac{d}{\gamma}}$, $\sigma \leftarrow \frac{1}{2q}\sqrt{d\gamma}$, $\theta \leftarrow 1 - \frac{1}{d+q\sqrt{d}/\sqrt{\gamma}}$
5: $w[0] = \mathbf{0}^{\mathrm{T}}$, $\eta[-1] = \eta[0] = \mathbf{1}^{\mathrm{T}}/n_1$, $\xi[-1] = \xi[0] = \mathbf{1}^{\mathrm{T}}/n_2$

Algorithm 2 Update Rules of Saddle-SVC

1: Pick an index i^* in $[d]$ uniformly at random
2: $\delta_{i^*}^+ \leftarrow \langle X_{i^*}^+, \eta[t] + \theta(\eta[t] - \eta[t-1])\rangle$, $\delta_{i^*}^- \leftarrow \langle X_{i^*}^-, \xi[t] + \theta(\xi[t] - \xi[t-1])\rangle$
3: $\forall i \in [d]$, $w_i[t+1] \leftarrow \begin{cases} (w_i[t] + \sigma(\delta_i^+ - \delta_i^-))/(\sigma+1), & \text{if } i = i^* \\ w_i[t], & \text{if } i \neq i^* \end{cases}$
4: $\eta[t+1] \leftarrow \arg\min_{\eta \in \mathcal{S}_1} \{\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X^+ \eta + \frac{\gamma}{d} H(\eta) + \frac{1}{\tau} V_{\eta[t]}(\eta)\}$
5: $\xi[t+1] \leftarrow \arg\min_{\xi \in \mathcal{S}_2} \{-\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X^- \xi + \frac{\gamma}{d} H(\xi) + \frac{1}{\tau} V_{\xi[t]}(\xi)\}$

For example, $w[0]$, $\eta[0], \xi[0]$ are the initial value of w, η, ξ and are defined in line 5 of Algorithm 1.

Update Rules: In order to unify HM-Saddle and ν-Saddle in the same framework, we use $(\mathcal{S}_1, \mathcal{S}_2)$ to represent the domains $(\Delta_{n_1}, \Delta_{n_2})$ in HM-Saddle (see formula (3)) or $(\mathcal{D}_{n_1}, \mathcal{D}_{n_2})$ in ν-Saddle (see formula (7)).

Generally speaking, the update rules alternatively maximize the objective with respect to w and minimize with respect to η and ξ. See the details in Algorithm 2.

Firstly, we update w according to line 3 in Algorithm 2. It is equivalent to a variant of the proximal coordinate gradient method with l_2-norm regularization as follows.

$$w_{i^*}[t+1] = \arg\max_{w_{i^*}} -\big\{ -(\delta_{i^*}^+ - \delta_{i^*}^-) w_{i^*} + w_{i^*}^2/2 + (w_{i^*} - w_{i^*}[t])^2/2\sigma \big\} \tag{9}$$

We briefly explain the intuition of (9). Note that the term $(\delta_{i^*}^+ - \delta_{i^*}^-)$ in (9) can be considered as the term $\langle X_{i^*}^+, \eta[t]\rangle - \langle X_{i^*}^-, \xi[t]\rangle$ adding extra momentum terms $\langle X_{i^*}^+, \theta(\eta[t] - \eta[t-1])\rangle$ and $-\langle X_{i^*}^-, \theta(\xi[t] - \xi[t-1])\rangle$ for dual variable $\eta[t]$ and $\xi[t]$ respectively (see line 2 in Algorithm 2). Further, $(\langle X_{i^*}^+, \eta[t]\rangle - \langle X_{i^*}^-, \xi[t]\rangle) w_{i^*} - w_{i^*}^2/2$ is the term in the objective function (4) and (8) which are related to w. The $(w_{i^*} - w_{i^*}[t])^2/2)$ is the l_2-norm regularization term.

Moreover, rather than update the whole w vector, randomly selecting one dimension $i^* \in [d]$ and updating the corresponding w_{i^*} in each iteration can reduce the runtime per round.

The update rules for η and ξ are listed in line 4 and 5 in Algorithm 2, which are the proximal gradient method with a Bergman divergence regularization $V_x(y) = H(y) - \langle \nabla H(x), y - x\rangle - H(x)$. Similar to $(\delta_{i^*}^+ - \delta_{i^*}^-)$ in (9), we also add a momentum term $d(w[t+1] - w[t])$ for primal variable w when updating η and ξ.

Projection Methods: However, the update rules for η and ξ are implicit update rules. We need to show that we can solve the corresponding optimization problems in line 4 and 5 of Algorithm 2 efficiently. In fact, for both HM-Saddle and ν-Saddle, we can obtain explicit expressions of these two optimization problems using the method of Lagrange multipliers.

First, we can solve the optimization problem for HM-Saddle (line 4 and 5 of Algorithm 2) directly, and the explicit expressions for η and ξ are as follows. The proof can be found in the full version.

▶ **Lemma 7** (Update Rules of HM-Saddle). *For linearly separable cases, the update rules in line 4 and 5 of Algorithms 2 is equivalent to*

$$\begin{aligned} \eta_i[t+1] &\leftarrow \Phi(\eta_i[t], X^+)/Z^+, \quad \forall i \in [n_1], \\ \xi_j[t+1] &\leftarrow \Phi(\xi_j[t], X^-)/Z^-, \quad \forall j \in [n_2], \end{aligned} \tag{10}$$

where Z^+ and Z^- are normalizers that ensures $\sum_i \eta_i[t+1] = 1$ and $\sum_j \xi_j[t+1] = 1$, and

$$\Phi(\lambda_i, X) = \exp\left\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\lambda_i - y_i \cdot \langle w[t] + d(w[t+1] - w[t], X_{\cdot i})\rangle)\right\} \tag{11}$$

Note that the factors Z^+ and Z^- are used to project the value $\Phi(\eta_i[t], X^+)$ and $\Phi(\xi_j[t], X^-)$ to the domains Δ_{n_1} and Δ_{n_2}. The above update rules of η and ξ can be also considered as the multiplicative weight update method (see [4]).

Next, we consider ν-Saddle. Compared to HM-Saddle, ν-Saddle has extra constraints that $\eta_i, \xi_j \le \nu$. Thus, we need another projection process to ensure that $\eta[t+1]$ and $\xi[t+1]$ locate in domain $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. For convenience, we only present the projection for η by the following Lemma 8. The projection for ξ is similar. Due to the space limit, we defer the proof of Lemma 8 to the full version.

▶ **Lemma 8** (Update Rules of ν-Saddle). *The following three update rules are equivalent.*

Rule 1:

$$\eta[t+1] := \arg\min_{\eta \in \mathcal{D}_{n_1}} \left\{\frac{1}{d}(w[t] + d(w[t+1] - w[t]))^{\mathrm{T}} X\eta + \frac{\gamma}{d}H(\eta) + \frac{1}{\tau}V_{\eta[t]}(\eta)\right\}$$

Rule 2:

- *Step 1:*

$$\eta_i := Z^{-1}\exp\left\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\eta_i[t] - \langle w[t] + d(w[t+1] - w[t]), X_{\cdot i}\rangle)\right\}$$

  for each $i \in [n_1]$, where Z ensures $\sum_i \eta_i = 1$.
- *Step 2:*

$$\begin{aligned} &\textbf{while} \quad \varsigma := \textstyle\sum_{\eta_i > \nu}(\eta_i - \nu) \neq 0 : \\ &\quad \Omega = \textstyle\sum_{\eta_i < \nu} \eta_i \\ &\quad \forall i, \quad \textbf{if } \eta_i \ge \nu, \quad \textbf{then } \eta_i[t+1] = \nu \\ &\quad \forall i, \quad \textbf{if } \eta_i < \nu, \quad \textbf{then } \eta_i[t+1] = \eta_i(1 + \varsigma/\Omega) \end{aligned} \tag{12}$$

Rule 3:

- *Step 1:*

$$\eta_i = Z^{-1}\exp\left\{(\gamma + d\tau^{-1})^{-1}(d\tau^{-1}\log\eta_i[t] - \langle w[t] + d(w[t+1] - w[t]), X_{\cdot i}\rangle)\right\}$$

  for each $i \in [n_1]$, where Z ensures $\sum_i \eta_i = 1$.
- *Step 2: Sort η_i by the increasing order. W.l.o.g., assume that $\eta_1, \ldots, \eta_{n_1}$ is in increasing order. Define $\varsigma_i = \sum_{j \ge i}(\eta_j - \nu)$ and $\Omega_i = \sum_{j<i}\eta_j$. Find the largest index $i^* \in [n]$ such that $\varsigma_{i^*} \ge 0$ and $\eta_{i^*-1}(1 + \varsigma_{i^*}/\Omega_{i^*}) < \nu$ by binary search.*

- *Step 3:*

$$\forall i, \eta_i[t+1] = \begin{cases} \eta_i(1+\varsigma_{i^*}/\Omega_{i^*}), & \text{if } i < i^* \\ \nu, & \text{if } i \geq i^* \end{cases}$$

We use Rule 2 when $1/\nu$ is constant. Note that there are at most $1/\nu$ (a constant) entries η_i of value ν during the whole projection process. In each iteration, there must be at least 1 more entry $\eta_i = \nu$ since we make all entries $\eta_j > \nu$ equal to ν after the iteration. Thus, the number of iterations in (12) is at most $1/\nu$. By (12), we project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. Thus, we need $O(n/\nu)$ time to compute $\eta[t+1]$. Since we assume ν is a constant, it only costs linear time.

When ν is extremely small, we use Rule 3 to project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$ respectively. It takes $O(n \log n)$ time because of sorting. Finally, we give our main theorem for our algorithm as follows. See the proof in the full version.

▶ **Theorem 9.** *Algorithm 2 computes $(1-\epsilon)$-approximate solutions for HM-Saddle and ν-Saddle by $\tilde{O}(d + \sqrt{d/\epsilon\beta})$ iterations. Moreover, it takes $O(n)$ time for each iteration.*

Combining with Lemmas 2, 3 and 5, we obtain $(1-\epsilon)$-approximate solutions for C-Hull and RC-Hull problems. Hence by strong duality, we obtain $(1-\epsilon)$-approximations for hard-margin SVM and ν-SVM in $\tilde{O}(n(d + \sqrt{d/\epsilon\beta}))$ time.

▶ **Theorem 10.** *A $(1-\epsilon)$-approximation for either hard-margin SVM or ν-SVM can be computed in $\tilde{O}(n(d + \sqrt{d/\epsilon\beta}))$ time.*

4 Distributed SVM

Server and Clients Model: We extend Saddle-SVC to the distributed setting and call it Saddle-DSVC. We consider the popular distributed setting: the *server* and *clients* model. Denote the server by S. Let $\mathcal{C}$ be the set of clients and $|\mathcal{C}| = k$. We use the notation $C.\alpha$ to represent any variable α saved in client C and use $S.\alpha$ to represent a variable α saved in the server.

First, we initialize some parameters in each client as the pre-processing step in Section 3. Each client maintains the same random diagonal matrix $D_{d\times d}$ and the total number of points in each type (i.e, $|\mathcal{P}| = n_1$ and $|\mathcal{Q}| = n_2$).[5] Moreover, each client C applies a Hadamard transformation to its own data and initialize the partial probability vectors $C.\eta$ and $C.\xi$ for its own points.

Formally speaking, assume there are m_1 points $x_1^+, x_2^+, \ldots, x_{m_1}^+$ and m_2 points $x_1^-, x_2^-, \ldots, x_{m_2}^-$ maintained in C. We use $\mathbf{1}^m$ to denote a vector with all components being 1. The initialization is as follows.

$$C.X^+ = WD \cdot [x_1^+, x_2^+, \ldots, x_{m_1}^+],\ C.\eta[-1] = C.\eta[0] = n_1^{-1}\mathbf{1}^{m_1}$$
$$C.X^- = WD \cdot [x_1^-, x_2^-, \ldots, x_{m_2}^-],\ C.\xi[-1] = C.\xi[0] = n_2^{-1}\mathbf{1}^{m_2}$$

We first consider HM-Saddle. The interaction between clients and the server can be divided into three rounds in each iteration.

1. In the first round, the server randomly chooses a number $i^* \in [d]$ and broadcasts i^* to all clients. Each client computes $C.\delta_{i^*}^+$ and $C.\delta_{i^*}^-$ and sends them back to the server.

[5] It can be realized using $O(k)$ communication bits.

2. In the second round, the server sums up all $C.\delta_{i^*}^+$ and $C.\delta_{i^*}^-$ and computes $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$. We can see that $S.\delta_{i^*}^+$ (resp. $S.\delta_{i^*}^-$) is exactly $\delta_{i^*}^+$ (resp. $\delta_{i^*}^-$) in Algorithm 2. The server broadcasts $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$ to all clients. By $S.\delta_{i^*}^+$ and $S.\delta_{i^*}^-$, each client updates w individually. Moreover, each client $C \in \mathcal{C}$ updates its own $C.\eta$ and $C.\xi$ according to the new directional vector w. In order to normalize the probability vectors η and ξ, each client sends the summation $C.Z^+$ and $C.Z^-$ to the server.
3. In the third round, the server computes $(S.Z^+, S.Z^-) \leftarrow \sum_{C\in\mathcal{C}}(C.Z^+, C.Z^-)$ and broadcasts to all clients the normalization factors $S.Z^+$ and $S.Z^-$. Finally, each client updates its partial probability vector $C.\eta$ and $C.\xi$ based on the normalization factors.

As we discuss in Section 3, for ν-Saddle, we need another $O(1/\nu)$ rounds to project η and ξ to the domains $\mathcal{D}_{n_1}$ and $\mathcal{D}_{n_2}$.

4. Each client computes $C.\varsigma^+, C.\varsigma^-$ and $C.\Omega^+$, $C.\Omega^-$ according to (12) and sends them to the server. The server sums up all $C.\varsigma^+, C.\varsigma^-, C.\Omega^+, C.\Omega^-$ respectively and gets $S.\varsigma^+, S.\varsigma^-$, $S.\Omega^+, S.\Omega^-$. If both $S.\varsigma^+$ and $S.\varsigma^-$ are zeros, the server stops this iteration. Otherwise, the server broadcasts to all clients the factors $S.\varsigma^+, S.\varsigma^-, S.\Omega^+, S.\Omega^-$. All clients update their $C.\eta$ and $C.\xi$ according to (12) and repeat Step 4 again.

We give the pseudocode in the full version. Note that all clients in Saddle-DSVC get the same $w[t]$ in each iteration as the $w[t]$ in Saddle-SVC. Hence Saddle-DSVC has the same rate of convergence as Saddle-SVC. Finally, after $T = \tilde{O}(d + \sqrt{d/\epsilon})$ iterations (see Theorem 9) , all clients compute the same $(1-\epsilon)$-approximate solution $w = w[T]$ for SVM. W.l.o.g, let the first client send w to the server. Based on the w (at most $O(n)$ more communication cost), the server can compute the offset b, the margin for hard-margin SVM and the objective value for the ν-SVM.

Communication Complexity of Saddle-DSVC: Note that in each iteration, the server and clients interact three times for hard-margin SVM and $O(1/\nu)$ times for ν-SVM. Thus, the communication cost of each iteration is $O(k)$. By Theorem 9, it takes $\tilde{O}(d+\sqrt{d/\epsilon})$ iterations. Thus, we summarize the following theorem.

▶ **Theorem 11.** *The communication cost of Saddle-DSVC is* $\tilde{O}(k(d + \sqrt{d/\epsilon}))$.

Liu et al. [26] prove that the lower bound of the communication cost for distributed SVM is $\Omega(k \min\{d, 1/\epsilon\})$.

▶ **Theorem 12** (Theorem 6 in [26])**.** *Consider a set of d-dimension points distributed at k clients. The communication cost to achieve a $(1-\epsilon)$-approximation of the distributed SVM problem is at least $\Omega(k \min\{d, 1/\epsilon\})$ for any $\epsilon > 0$.*

If $d = \Theta(1/\epsilon)$, the communication lower bound is $\Omega(k(d + \sqrt{d/\epsilon}))$ which matches the communication cost of Saddle-DSVC.

References

1 Nir Ailon and Bernard Chazelle. Faster dimension reduction. *Communications of the ACM*, 53(2):97–104, 2010.

2 Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *STOC 16*, 2016.

3 Zeyuan Allen-Zhu, Zhenyu Liao, and Yang Yuan. Optimization algorithms for faster computational geometry. In *LIPIcs*, volume 55, 2016.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

5 Kristin P Bennett and Erin J Bredensteiner. Duality and geometry in svm classifiers. In *ICML*, pages 57–64, 2000.

6 Marshall W. Bern and David Eppstein. Optimization over zonotopes and training support vector machines. In *WADS*, pages 111–121, 2001.

7 Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

8 Kenneth L Clarkson, Elad Hazan, and David P Woodruff. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):23, 2012.

9 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

10 DJ Crisp and CJC Burges. A geometry interpretation of μ-svm classifiers. *NIPS*, pages 244–251, 2000.

11 John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *JMLR*, 10(Dec):2899–2934, 2009.

12 Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. Consensus-based distributed support vector machines. *JMLR*, 11(May):1663–1707, 2010.

13 Vojtěch Franc and Soeren Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML 08*, pages 320–327. ACM, 2008.

14 Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

15 Bernd Gärtner and Martin Jaggi. Coresets for polytope distance. In *SOCG 09*, pages 33–42. ACM, 2009.

16 Elmer G Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 4(1):61–80, 1966.

17 Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*, volume 17, 2004.

18 Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI*, pages 836–841, 2007.

19 Elad Hazan, Tomer Koren, and Nati Srebro. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, pages 1233–1241, 2011.

20 Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *ICML 08*, pages 408–415. ACM, 2008.

21 Thorsten Joachims. Making large-scale svm learning practical. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1998.

22 Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

23 Anatoli Juditsky, Fatma Kılınç Karzan, and Arkadi Nemirovski. Randomized first order algorithms with applications to ℓ_1-minimization. *Mathematical Programming*, 142(1-2):269–310, 2013.

24 Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

25 Eyal Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997.

26 Yangwei Liu, Hu Ding, Ziyun Huang, and Jinhui Xu. Distributed and robust support vector machine. In *LIPIcs*, volume 64, 2016.

27 Jorge López and José R Dorronsoro. Linear convergence rate for the mdm algorithm for the nearest point problem. *Pattern Recognition*, 48(4):1510–1522, 2015.

28 Yumao Lu, Vwani Roychowdhury, and Lieven Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *IEEE Transactions on Neural Networks*, 19(7):1167–1178, 2008.

29 BF Mitchell, Vladimir Fedorovich Dem'yanov, and VN Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM Journal on Control*, 12(1):19–26, 1974.

30 A Navia-Vazquez, D Gutierrez-Gonzalez, Emilio Parrado-Hernández, and JJ Navarro-Abellan. Distributed support vector machines. *IEEE Trans. Neural Networks*, 17(4):1091–1097, 2006.

31 Yu Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16(1):235–249, 2005.

32 Alon Orlitsky and Abbas El Gamal. Average and randomized communication complexity. *IEEE Transactions on Information Theory*, 36(1):3–16, 1990.

33 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12(Oct):2825–2830, 2011.

34 John C Platt. 12 fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

35 Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

36 Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Math. Program.*, 127(1):3–30, 2011.

37 Alexander J Smola, SVN Vishwanathan, Quoc V Le, et al. Bundle methods for machine learning. In *NIPS*, volume 20, pages 1377–1384, 2007.

38 Ivor W Tsang, Andras Kocsor, and James T Kwok. Simpler core vector machines with enclosing balls. In *ICML 07*, pages 911–918. ACM, 2007.

39 Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *JMLR*, 6(Apr):363–392, 2005.

40 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC 79*, pages 209–213. ACM, 1979.

41 Caoxie Zhang, Honglak Lee, and Kang G Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Artificial Intelligence and Statistics*, pages 1398–1406, 2012.

42 Yuchen Zhang and Xiao Lin. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *ICML*, pages 353–361, 2015.

43 Ji Zhu, Saharon Rosset, Robert Tibshirani, and Trevor J Hastie. 1-norm support vector machines. In *NIPS*, pages 49–56, 2004.

Lower Bounds on Sparse Spanners, Emulators, and Diameter-reducing shortcuts*

Shang-En Huang
University of Michigan, USA
sehuang@umich.edu

Seth Pettie
University of Michigan, USA
pettie@umich.edu

Abstract

We prove better lower bounds on additive spanners and emulators, which are lossy compression schemes for *undirected* graphs, as well as lower bounds on *shortcut sets*, which reduce the diameter of *directed* graphs. We show that any $O(n)$-size shortcut set cannot bring the diameter below $\Omega(n^{1/6})$, and that any $O(m)$-size shortcut set cannot bring it below $\Omega(n^{1/11})$. These improve Hesse's [16] lower bound of $\Omega(n^{1/17})$. By combining these constructions with Abboud and Bodwin's [1] edge-splitting technique, we get additive stretch lower bounds of $+\Omega(n^{1/13})$ for $O(n)$-size spanners and $+\Omega(n^{1/18})$ for $O(n)$-size emulators. These improve Abboud and Bodwin's $+\Omega(n^{1/22})$ lower bounds.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases additive spanners, emulators, shortcutting directed graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.26

Acknowledgements Thanks to the reviewers who point out the detailed comparison between spanners and emulators. We also thank the reviewers who carefully and explicitly verified our constructions, and give us some insightful comments.

1 Introduction

A *spanner* of an undirected unweighted graph $G = (V, E)$ is a subgraph H that approximates the distance function of G up to some *stretch*. An *emulator* for G is defined similarly, except that H need not be a subgraph, and may contain *weighted* edges. In this paper we consider only *additive* stretch functions:

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq \text{dist}_G(u, v) + \beta,$$

where β may depend on n.

Graph compression schemes (like spanners and emulators) are related to the problem of *shortcutting* digraphs to reduce diameter, inasmuch as lower bounds for both objects are constructed using the same suite of techniques. These lower bounds begin from the construction of graphs in which numerous pairs of vertices have shortest paths that are *unique*, *edge-disjoint*, and relatively *long*. Such graphs were independently discovered by Alon [4], Hesse [16], and Coppersmith and Elkin [12]; see also [1, 2]. Given such a "base graph," derived graphs can be obtained through a variety of graph products such as the

* This work was supported by NSF grants CCF-1514383 and CCF-1637546.

Table 1 Upper and Lower bounds on shortcutting sets. The lower bounds are existential, and independent of computation time.

Citation	Shortcut Set Size	Diameter	Computation Time
Folklore/trivial	$O(n)$	$\tilde{O}(\sqrt{n})$	$O(m\sqrt{n})$
	$O(m)$	$\tilde{O}(n/\sqrt{m})$	$O(m^{3/2})$
Fineman [15]	$\tilde{O}(n)$	$\tilde{O}(n^{2/3})$	$\tilde{O}(m)$
Hesse [16]	$O(mn^{1/17})$	$\Omega(n^{1/17})$	—
new	$O(n)$	$\Omega(n^{1/6})$	—
	$O(m)$	$\Omega(n^{1/11})$	—

alternation product discovered independently by Hesse [16] and Abboud and Bodwin [1] and the *substitution* product used by Abboud and Bodwin [1] and developed further by Abboud, Bodwin, and Pettie [2].

In this paper we apply the techniques developed in [4, 16, 12, 1, 2] to obtain better lower bounds on shortcutting sets, additive spanners, and additive emulators.

Shortcutting Sets

Let $G = (V, E)$ be a directed graph and $G^* = (V, E^*)$ its transitive closure. The *diameter* of a digraph G is the maximum of $\text{dist}_G(u, v)$ over all pairs $(u, v) \in E^*$. Thorup [20] conjectured that it is possible to reduce the diameter of any digraph to $\text{poly}(\log n)$ by adding a set $E' \subseteq E^*$ of at most $m = |E|$ *shortcuts*, i.e., $G' = (V, E \cup E')$ would have diameter $\text{poly}(\log n)$. This conjecture was confirmed for a couple special graph classes [20, 21], but refuted in general by Hesse [16], who exhibited a graph with $m = \Theta(n^{19/17})$ edges and diameter $\Theta(n^{1/17})$ such that any diameter-reducing shortcutting requires $\Omega(mn^{1/17})$ shortcuts. More generally, there exist graphs with $m = n^{1+\epsilon}$ edges and diameter n^δ, $\delta = \delta(\epsilon)$, that require $\Omega(n^{2-\epsilon})$ shortcuts to make the diameter $o(n^\delta)$; see Abboud, Bodwin, and Pettie [2, §6] for an alternative proof of this result.

On the upper bound side, it is trivial to reduce the diameter to $\tilde{O}(\sqrt{n})$ with $O(n)$ shortcuts or diameter $\tilde{O}(n/\sqrt{m})$ with $O(m)$ shortcuts.[1] Unfortunately, the trivial shortcutting schemes are not efficiently constructible in near-linear time. In some applications of shortcuttings, efficiency of the construction is just as important as reducing the diameter. For example, a longstanding problem in parallel computing is to *simultaneously* achieve time and work efficiency in computing reachability.[2] Very recently, Fineman [15] proved that an $\tilde{O}(n)$-size shortcut set can be computed in near-optimal work $\tilde{O}(m)$ (and $\tilde{O}(n^{2/3})$ parallel time) that reduces the diameter to $\tilde{O}(n^{2/3})$.

In this paper we prove that $O(n)$-size shortcut sets cannot reduce the diameter below $\Omega(n^{1/6})$, and that $O(m)$-size shortcut sets cannot reduce it below $\Omega(n^{1/11})$. See Table 1.

Additive Spanners

Additive spanners with constant stretches were discovered by Aingworth, Checkuri, Indyk, and Motwani [3] (see also [13, 14, 5, 18]), Chechik [11], and Baswana, Kavitha, Mehlhorn, and Pettie [5] (see also [23, 18]). The sparsest of these [5] has size $O(n^{4/3})$ and stretch

[1] Pick a set S of $\sqrt{n}$ or $\sqrt{m}$ vertices uniformly at random, and include $S^2 \cap E^*$ as shortcuts.
[2] This is the notorious *transitive closure bottleneck*.

Table 2 Upper and lower bounds on additive spanners.

Citation	Spanner Size	Additive Stretch	Remarks
Aingworth, Chekuri, Indyk, and Mowani [3]	$O(n^{3/2})$	2	See also [13, 14, 5, 18]
Chechik [11]	$\tilde{O}(n^{7/5})$	4	
Baswana, Kavitha, Mehlhorn, and Pettie [5]	$O(n^{4/3})$	6	See also [23, 18]
Pettie [19]	$O(n^{1+\epsilon})$	$O(n^{9/16-7\epsilon/8})$	$0 \le \epsilon$
Chechik [11]	$O(n^{20/17+\epsilon})$	$O(n^{4/17-3\epsilon/2})$	$0 \le \epsilon$
Bodwin and Williams [9]	$O(n^{1+\epsilon})$	$O(n^{1/2-\epsilon/2})$ $O(n^{2/3-5\epsilon/3})$	$0 \le \epsilon$
Bodwin and Williams [8]	$O(n^{1+o(1)+\epsilon})$	$O(n^{3/7-\epsilon})$	$0 \le \epsilon \le 6/49$
		$O(n^{3/5-12\epsilon/5})$	$6/49 \le \epsilon \le 2/13$
		$O(n^{3/7-9\epsilon/7})$	$2/13 \le \epsilon < 1/3$
Abboud and Bodwin [1]	$O(n^{4/3-\epsilon})$	$\Omega(n^{\delta})$	$\delta = \delta(\epsilon)$
	$O(n)$	$\Omega(n^{1/22})$	
new	$O(n)$	$\Omega(n^{1/13})$	

$+6$. Abboud and Bodwin [1] showed that the 4/3 exponent could not be improved, in the sense that any $+n^{o(1)}$ spanner has size $\Omega(n^{4/3-o(1)})$, and that any $\Omega(n^{4/3-\epsilon})$-size spanner has additive stretch $+\Omega(n^{\delta})$, $\delta = \delta(\epsilon)$. On the upper bound side, Pettie [19] showed that $O(n)$-size spanners could have additive stretch $+\tilde{O}(n^{9/16})$, and Bodwin and Williams [8] improved this to $O(\sqrt{n})$ for $O(n)$-size spanners and $O(n^{3/7})$ for $O(n^{1+o(1)})$-size spanners. Abboud and Bodwin [1] extended their lower bound to $O(n)$-size spanners, showing that they require stretch $+\Omega(n^{1/22})$. Using our lower bound for shortcuttings as a starting place, we improve [1] by giving an $+\Omega(n^{1/13})$ stretch lower bound for $O(n)$-size spanners. See Table 2.

Additive Emulators

Dor, Halperin, and Zwick [13] were the first to explicitly define the notion of an *emulator*, and gave a $+4$ emulator with size $O(n^{4/3})$. Abboud and Bodwin's [1] lower bound applies to emulators, i.e., we cannot go below the 4/3 threshold without incurring polynomial additive stretch. Bodwin and Williams [9, 8] pointed out that some spanner construtions [5] imply emulator bounds, and gave new constructions of emulators with size $O(n)$ and stretch $+O(n^{1/3})$, and with size $O(n^{1+o(1)})$ and stretch $+O(n^{3/11})$.[3] Here we observe that Pettie's [19] $+\tilde{O}(n^{9/16})$ spanner, when turned into an $O(n)$-size emulator, has stretch $+\tilde{O}(n^{1/4})$, which is slightly better than the linear size emulators found in [5, 9, 8]. We improve Abboud and Bodwin's [1] lower bound and show that any $O(n)$-size emulator has additive stretch $+\Omega(n^{1/18})$. See Table 3.

Our emulator lower bounds are polynomially weaker than the spanner lower bounds. Although neither bound is likely sharp, this difference reflects the rule that emulators are probably more powerful than spanners. For example, at sparsity $O(n^{4/3})$, the best known emulators [13] are slightly better than spanners [5]. Below the 4/3 threshold the best *sublinear*

[3] This last result is a consequence of [8, Thm. 5] and the fact that any pair set $P \subset V^2$ has a pair-wise emulator with size $|P|$.

Table 3 Upper and lower bounds on additive emulators. Emulators with sublinear additive stretch [22, 17, 2] are not shown.

Citation	Emulator Size	Additive Stretch	Remarks
Aingworth, Chekuri, Indyk, and Mowani [3]	$O(n^{3/2})$	2	See also [13, 14, 5, 18]
Dor, Halperin, and Zwick [13]	$O(n^{4/3})$	4	
Baswana, Kavitha, Mehlhorn and Pettie [5]	$O(n^{1+\epsilon})$	$O(n^{1/2-3\epsilon/2})$	(not claimed in [5])
Bodwin and Williams [9]	$O(n^{1+\epsilon})$	$O(n^{1/3-2\epsilon/3})$	
Bodwin and Williams [8]	$O(n^{1+o(1)+\epsilon})$	$O(n^{3/11-9\epsilon/11})$	(conseq. of [8, Thm. 5])
Pettie [19]	$O(n^{1+\epsilon})$	$\tilde{O}(n^{1/4-3\epsilon/4})$	(not claimed in [19])
Abboud and Bodwin [1]	$O(n)$	$\Omega(n^{1/22})$	
new	$O(n)$	$\Omega(n^{1/18})$	

additive emulators [22, 17] have size $O(n^{1+\frac{1}{2^{k+1}-1}})$ and stretch function $d + O(d^{1-1/k})$.[4] Abboud, Bodwin, and Pettie [2] showed that this tradeoff is optimal for emulators, but the best sublinear additive spanners [19, 11] are polynomially worse.

There are a certain range of parameters where the emulators are polynomially better than the spanners. On pairwise distance preservers, Bodwin [7] showed that whenever $\omega(n^{1/2}) = |P| = o(n^{2-o(1)})$, any pairwise distance preserver has an $\omega(n + |P|)$ lower bound, creating a gap comparing to an $\Theta(|P|)$ emulator.

There is also another situation where emulators are provable superior: a source-wise distance preserver for $S \subset V$ maintains distances between S-vertices without stretch. A trivial source-wise emulator has size $|S|^2$, e.g., $O(n)$ for $|S| = \sqrt{n}$, but in [12, 7] source-wise spanners with size $O(n)$ only exist for $|S| = O(n^{1/4})$.

Outline

In Section 2 we present diameter lower bounds for shortcut sets of size $O(n)$ and $O(m)$. Section 3 modifies the construction to give lower bounds on additive spanners and additive emulators. We conclude with some remarks in Section 4.

2 Lower Bounds on Shortcutting Digraphs

2.1 Using $O(n)$ Shortcuts

▶ **Theorem 1.** *There exists a directed graph G with n vertices, such that for any shortcut set E' with size $O(n)$, the graph $(V, E \cup E')$ has diameter $\Omega(n^{1/6})$.*

The remainder of Section 2.1 constitutes a proof of Theorem 1. We begin by defining the vertex set and edge set of G, and its *critical pairs*.

Vertices

The vertex set of G is partitioned into $D + 1$ *layers* numbered 0 through D. Define $B_d(\rho)$ to be the set of all lattice points in $\mathbb{Z}^d$ within Euclidean distance ρ of the origin. Here we treat

[4] I.e., vertices initially at distance d are stretched to $d + O(d^{1-1/k})$.

d as a constant. For each $k \in \{0, \ldots, D\}$, layer-k vertices are identified with lattice points in $B_d(R + kr)$, where r, R are parameters of the construction. A vertex can be represented by a pair (a, k), where $a \in B_d(R + rk)$. We want the size of all layers to be the same, up to a constant factor. To that end we fix $R = drD$, so the total number of vertices is

$$\begin{aligned} n &\approx \eta_d R^d \left(1^d + \left(1 + \frac{r}{R}\right)^d + \cdots + \left(1 + \frac{rD}{R}\right)^d \right) \\ &= \eta_d R^d \left(1^d + \left(1 + \frac{1}{dD}\right)^d + \cdots + \left(1 + \frac{1}{d}\right)^d \right) \;=\; \Theta\left(R^d D\right) \quad \text{(By definition of } R\text{)} \end{aligned}$$

where $\eta_d = \frac{1}{\sqrt{2\pi d}} \left(\frac{2\pi e}{d}\right)^{d/2}$ is the ratio of volume between a d-dimentional ball and a d-dimentional cube.

Edges

Define $\mathcal{V}_d(r)$ to be the set of all lattice points at the corners of the convex hull of $B_d(r)$. We treat elements of $\mathcal{V}_d(r)$ as vectors. For each layer-k vertex (a, k), $k \in \{0, \ldots, D-1\}$, and each vector $v \in \mathcal{V}_d(r)$, we include a directed edge $((a, k), (a + v, k + 1))$. All edges in G are of this form.

Critical Pairs

The critical pair set is defined to be

$$P = \{((a, 0), (a + Dv, D)) \mid a \in B_d(R) \text{ and } v \in \mathcal{V}_d(r)\}$$

Each such pair has a corresponding path of length D, namely $(a, 0) \to (a + v, 1) \to \cdots \to (a + Dv, D)$. Lemma 2 shows that this path is unique. It was first proved by Hesse [16] and independently by Coppersmith and Elkin [12]. (Both proofs are inspired by Behrend's [6] construction of arithmetic progression-free sets, which uses ℓ_2 balls rather than convex hulls.)

▶ **Lemma 2.** *(cf. [16, 12]) The set of critical pairs P have the following properties:*

- *For all $(x, y) \in P$, there is a unique path from x to y in G.*
- *For any two distinct pairs (x_1, y_1) and $(x_2, y_2) \in P$, their unique paths share no edge and at most one vertex.*
- $|P| = \Theta(R^d r^d \frac{d-1}{d+1})$.

Proof.

- Let $x = (a, 0)$ and $v \in \mathcal{V}_d(r)$ be the vector for which $y = (a + Dv, D)$. One path from x to y exists by construction. Let $\mathcal{V}_d(r) = \{v_1, v_2, \ldots, v_s\}$. Suppose there exists another path from x to y. It must have length D because all edges join consecutive layers. Every edge on this path corresponds to a vector v_i, which implies that Dv can be represented as a linear combination $k_1 v_1 + k_2 v_2 + \cdots + k_s v_s$, where $k_1 + \cdots + k_s = D$ and $k_i \geq 0$. This implies that v is a non-trivial convex combination of the vectors in $\mathcal{V}_d(r)$, which contradicts the fact that $\mathcal{V}_d(r)$ is a strictly convex set.
- Observe that any edge in the unique (x_1, y_1) path uniquely identifies both x_1 and y_1.
- $|P| = |B_d(R)| \cdot |\mathcal{V}_d(r)|$. From Bárány and Larman [10], for any constant dimension d, we have $|\mathcal{V}_d(r)| = \Theta(r^d \frac{d-1}{d+1})$. ◀

▶ **Lemma 3.** *Let E' be a shortcut set for $G = (V, E)$. If the diameter of $G' = (V, E \cup E')$ is strictly less than D, then $|E'| \geq |P|$.*

Proof. Every path in G' corresponds to some path in G. However, for pairs in P, there is only one path in G, hence, any shortcut in E' useful for a pair $(x, y) \in P$ must have both endpoints on the unique x-y path in G. By Lemma 2, two such paths for pairs in P share no common edges, hence each shortcut can only be useful for at most one pair in P. If $|E'| < |P|$ then some pair $(x, y) \in P$ must still be at distance D in G'. ◀

Proof of Theorem 1. By Lemma 3, if $|P| = \Omega(n)$, then any shortcut set that makes the diameter $< D$ has size $\Omega(n)$. In order to have $|P| = \Omega(n)$, it suffices to let $r^d \frac{d-1}{d+1} \geq D$. This implies $r \geq D^{\frac{d+1}{d(d-1)}}$. From the construction, by fixing d as a constant, we have

$$n \;=\; \Theta(R^d D) \;=\; \Theta((rD)^d D) \;=\; \Omega(D^{1+d+\frac{d+1}{d-1}}).$$

Therefore, the diameter is $D = O\left(n^{1/\left(1+d+\frac{d+1}{d-1}\right)}\right)$. We can maximize $D = \Theta(n^{1/6})$ in one of two ways, by setting $d = 2$, $r = \Theta(n^{1/4})$, and $R = \Theta(n^{5/12})$, or $d = 3$, $r = \Theta(n^{1/9})$, and $R = \Theta(n^{5/18})$. In either case, the construction leads to a graph with very similar structure: the number of vertices in each layer is $\Theta(n^{5/6})$, and the out degrees of each vertex are $\Theta(n^{1/6})$. ◀

▶ **Corollary 4.** *Fix an $\epsilon \in [0, 1)$ and let d be such that $\epsilon \in [0, \frac{d-1}{d+1}]$. There exists a directed graph G with n vertices, such that for any shortcut set E' with $O(n^{1+\epsilon})$ shortcuts, the graph $(V, E \cup E')$ has diameter $\Omega(n^{\left(1-\frac{d+1}{d-1}\epsilon\right)/\left(1+d+\frac{d+1}{d-1}\right)})$. In particular, by setting $d = 3$ the diameter lower bound becomes $\Omega(n^{\frac{1}{6}-\frac{1}{3}\epsilon})$.*

Proof. In order to have $|P| > n^{1+\epsilon}$, it suffices to let $r^d \frac{d-1}{d+1} \geq Dn^\epsilon$. Hence, we have

$$\begin{aligned} n^{1-\frac{d+1}{d-1}\epsilon} &= \Theta(R^d D n^{-\frac{d+1}{d-1}\epsilon}) \\ &= \Omega(r^d D^{1+d} n^{-\frac{d+1}{d-1}\epsilon}) && (R = \Theta(rD)) \\ &= \Omega(D^{1+d+\frac{d+1}{d-1}}) && (r^d \geq (Dn^\epsilon)^{\frac{d+1}{d-1}}) \end{aligned}$$

◀

2.2 Using $O(m)$ Shortcuts

Let $G_{(d,r,D)}$ denote the layered graph constructed in Section 2.1 with parameters d, D, r, and $R = drD$, and let P_G be its critical pair set. The total number of edges $m = \Theta(n|\mathcal{V}_d(r)|)$ is always larger than $|P_G| = \Theta(\frac{n}{D}|\mathcal{V}_d(r)|)$ by a factor of D. In order to get a lower bound for $O(m)$ shortcuts, we use a Cartesian product combining two such graphs layer by layer, forming a sparser graph. This transformation was discovered by Hesse [16] and rediscovered by Abboud and Bodwin [1].

Let $G_1 = G_{(d_1,r_1,D)}$ and $G_2 = G_{(d_2,r_2,D)}$ be two graphs with the same number of vertex layers $(D + 1)$. The product graph $G_1 \otimes G_2$ is defined below.

Vertices

The product graph has $2D + 1$ vertex layers numbered $0, \ldots, 2D$. The vertex set of layer i is $\{(x, y, i) \mid x \in B_{d_1}(R_1 + \lceil \frac{i}{2} \rceil r_1), y \in B_{d_2}(R_2 + \lfloor \frac{i}{2} \rfloor r_2)\}$. Since we set $R_j = d_j r_j D$, the total number of vertices is $\Theta\left(R_1^{d_1} R_2^{d_2} D\right)$.

Edges

Let (x, y, i) be a vertex in layer i. If i is even, then for every vector $v \in \mathcal{V}_{d_1}(r_1)$ we include an edge $((x, y, i), (x + v, y, i + 1))$. If i is odd, then for every vector $w \in \mathcal{V}_{d_2}(r_2)$, we include an edge $((x, y, i), (x, y + w, i + 1))$. The total number of edges in the product graph is then $\Theta\left(R_1^{d_1} R_2^{d_2} D \left(r_1^{d_1 \frac{d_1-1}{d_1+1}} + r_2^{d_2 \frac{d_2-1}{d_2+1}}\right)\right)$.

Critical Pairs

By combining two graphs, we are able to construct a larger set of critical pairs, as follows.

$$P = \{((a, b, 0), (a+Dv, b+Dw, 2D)) \mid a \in B_{d_1}(R_1), b \in B_{d_2}(R_2), v \in \mathcal{V}_{d_1}(r_1), w \in \mathcal{V}_{d_2}(r_2)\}$$

In other words, a pair in P can be viewed as the product of two pairs $((a, 0), (a+Dv, D)) \in P_{G_1}$ and $((b, 0), (b + Dw, D)) \in P_{G_2}$.

▶ **Lemma 5.** *For any $a \in B_{d_1}(R_1)$, $b \in B_{d_2}(R_2)$, $v \in \mathcal{V}_{d_1}(r_1)$ and $w \in \mathcal{V}_{d_2}(r_2)$, there is a unique path from $(a, b, 0)$ to $(a + Dv, b + Dw, 2D)$.*

Proof. Every path in $G_1 \otimes G_2$ from layer 0 to layer $2D$ corresponds to two paths from layers 0 to D in G_1 and G_2, respectively. It follows from Lemma 2 that

$$(a, b, 0) \to (a + v, b, 1) \to (a + v, b + w, 2) \to \cdots \to (a + Dv, b + Dw, 2D)$$

is a unique path in $G_1 \otimes G_2$. ◀

In $G_1 \otimes G_2$ it is no longer true that pairs in P have edge-disjoint paths. They may intersect at just one edge.

▶ **Lemma 6.** *Consider two pairs (x_1, y_1) and $(x_2, y_2) \in P$. Let P_1 and P_2 be the unique shortest paths in the combined graph from x_1 to y_1 and from x_2 to y_2. Then, $P_1 \cap P_2$ contains at most one edge.*

Proof. Any two *non-adjacent* vertices on the unique x_1-y_1 path uniquely identify x_1 and y_1. Thus, two such paths can intersect in at most 2 (consecutive) vertices, and hence one edge. ◀

▶ **Lemma 7.** *Let E' be a shortcut set on $G = (V, E)$. If the diameter of $(V, E \cup E')$ is strictly less than $2D$, then $|E'| \geq |P|$.*

Proof. Assume the diameter of $(V, E \cup E')$ is strictly less than $2D$. Every useful shortcut connects vertices that are at distance at least 2. By Lemma 6, such a shortcut can only be useful for one pair in P. Thus, if the diameter of $(V, E \cup E')$ is less than $2D$, $|E'| \geq |P|$. ◀

By construction, the size of $|P|$ is

$$|P| = \Theta\left(R_1^{d_1} R_2^{d_2} |\mathcal{V}_{d_1}(r_1)||\mathcal{V}_{d_2}(r_2)|\right) = \Theta\left(R_1^{d_1} R_2^{d_2} r_1^{d_1 \frac{d_1-1}{d_1+1}} r_2^{d_2 \frac{d_2-1}{d_2+1}}\right).$$

▶ **Theorem 8.** *There exists a directed graph G with n vertices and m edges such that for any shortcut set E' with size $O(m)$, the graph $(V, E \cup E')$ has diameter $\Omega(n^{1/11})$.*

Proof. If we set $|P| = \Omega(m)$, by Lemma 7, any shortcut set E' with $O(m)$ shortcuts has diameter $\Omega(D)$. In order to ensure $|P| = \Omega(m)$, it suffices to set $r_1^{d_1\frac{d_1-1}{d_1+1}} \geq r_2^{d_2\frac{d_2-1}{d_2+1}} \geq D$. Hence,

$$\begin{aligned} n &= \Theta(R_1^{d_1} R_2^{d_2} D) \\ &= \Theta\left(r_1^{d_1} r_2^{d_2} D^{d_1+d_2+1}\right) && (R_j = d_j r_j D) \\ &= \Omega\left(D^{\frac{d_1+1}{d_1-1}} D^{\frac{d_2+1}{d_2-1}} D^{d_1+d_2+1}\right) && \text{(plugging in relation between } r_j \text{ and } d_j, D) \\ &= \Omega\left(D^{\frac{d_1+1}{d_1-1}+\frac{d_2+1}{d_2-1}+d_1+d_2+1}\right) \end{aligned}$$

The exponent is minimized when d_1 and d_2 are either 2 or 3. By setting $d_1 = d_2 = 2$, we get $n = \Omega(D^{11})$ and hence $D = O(n^{1/11})$. In this construction we have $d_1 = d_2 = 2$, $D = \Theta(n^{1/11})$, $r_1 = r_2 = \Theta(n^{3/22})$ and $R_1 = R_2 = \Theta(n^{5/22})$. ◀

3 Lower Bounds on Additive Spanners and Emulators

3.1 $O(n)$-sized Spanners

▶ **Definition 9.** Let $G = (V, E)$ be an (unweighted) undirected graph. A subgraph $H = (V, E' \subseteq E)$ is said to be an *spanner with additive stretch* β if for any two vertices $x, y \in V$, $\text{dist}_G(x, y) \leq \text{dist}_H(x, y) \leq \text{dist}_G(x, y) + \beta$.

By combining the technique of Abboud and Bodwin [1] with the graphs constructed in Section 2.2, we obtain a substantially better lower bound on $O(n)$-size additive spanners.

▶ **Theorem 10.** *There exists an undirected graph G with n vertices, such that any spanner with $O(n)$ edges has $+\Omega(n^{1/13})$ additive stretch.*

In this section we regard $G_{(d,r,D)}$ to be an *undirected* graph. We begin with the undirected graph $G_0 = G_{(d_1,r_1,D)} \otimes G_{(d_2,r_2,D)}$, then modify it in the *edge expansion step* and the *clique replacement step* to obtain G.

The Edge Expansion Step

Every edge in G_0 is subdivided into D edges, yielding G_E. This step makes the graph very sparse since most of the vertices in G_E have degree 2.

The Clique Replacement Step

Consider a vertex u in G_E that comes from one of the interior layers of G_0, i.e., layers $1, \ldots, 2D-1$, not 0 or $2D$. Note that u has degree $\delta_1 + \delta_2$, with $\delta_1 = \Theta\left(r_1^{d_1\frac{d_1-1}{d_1+1}}\right)$ edges leading to the preceding layer and $\delta_2 = \Theta\left(r_2^{d_2\frac{d_2-1}{d_2+1}}\right)$ edges leading to the following layer (or vice versa). We replace each such u with a complete bipartite clique K_{δ_1,δ_2}, where each clique vertex becomes attached to one non-clique edge formerly attached to u. The final graph is denoted by G.

Critical Pairs

The set P of *critical pairs* for G is identical to the set of critical pairs for G_0. For each $(x, y) \in P$, the unique x-y path in G is called a *critical path*.

From the construction, the number of vertices in G is then

$$n = \Theta\left(R_1^{d_1} R_2^{d_2} D^2(\delta_1 + \delta_2)\right). \tag{1}$$

The number of edges in G is now

$$m = \Theta\left(R_1^{d_1} R_2^{d_2} D(D\delta_1 + D\delta_2 + \delta_1\delta_2)\right). \tag{2}$$

The size of P is

$$|P| = \Theta\left(R_1^{d_1} R_2^{d_2} \delta_1\delta_2\right). \tag{3}$$

Lemma 11 is key to relating the size of the spanner with the pair set P.

▶ **Lemma 11.** *Every clique edge belongs to at most one critical path.*

Proof. Every clique has δ_1 vertices on one side and δ_2 vertices on the other side. Each vertex on the δ_1 side corresponds to a vector $v \in \mathcal{V}_{d_1}(r_1)$ and each vertex on the δ_2 side corresponds to a vector $w \in \mathcal{V}_{d_2}(r_2)$. Each clique edge uniquely determines a pair of vectors (v, w), and hence exactly one critical pair in P. ◀

▶ **Lemma 12.** *Every spanner of G with additive stretch $+(2D-1)$ must contain at least $D|P|$ clique edges.*

Proof. For the sake of contradiction suppose there exists a spanner H containing at most $D|P| - 1$ clique edges. By the pigeonhole principle there exists a pair $(x, y) \in P$ such that at least D clique edges are *missing* in H.

Let $P_{(x,y)}$ be the unique shortest path from x to y in G, and let $P'_{(x,y)}$ be a shortest path from x to y in H. Since G_0 is formed from G by contracting all bipartite cliques and replacing subdivided edges with single edges, we can apply the same operations on $P'_{(x,y)}$ to get a path $P''_{(x,y)}$ in G_0. We now consider two cases:

- If $P''_{(x,y)}$ is the unique shortest path from x to y in G_0, then $P'_{(x,y)}$ suffers at least a $+2$ stretch on each of the D missing clique edges, so $|P'_{(x,y)}| \geq |P_{(x,y)}| + 2D$.
- If $P''_{(x,y)}$ is not the unique shortest path from x to y in G_0, then it must traverse at least two more edges than the shortest x-y path in G_0 (because G_0 is bipartite), each of which is subdivided D times in the formation of G. Thus $|P'_{(x,y)}| \geq |P_{(x,y)}| + 2D$.

In either case, $P'_{(x,y)}$ has at least $+2D$ additive stretch and H cannot be a $+(2D-1)$ spanner. ◀

Proof of Theorem 10. The goal is to have parameters set up so that $D|P| = \Omega(n)$, so that we can apply Lemma 12. Without loss of generality $\delta_1 \geq \delta_2$. By comparing (1) with (3), it suffices to set $\delta_1 \geq \delta_2 \geq D$. We can express the number of vertices in terms of D as follows:

$$\begin{aligned}
n &= \Theta\left(R_1^{d_1} R_2^{d_2} D^2 \delta_1\right) \\
&= \Omega\left((r_1 D)^{d_1} (r_2 D)^{d_2} D^3\right) && (\delta_1 \geq \delta_2 \geq D) \\
&= \Omega\left(\left(\delta_1^{\frac{d_1+1}{d_1(d_1-1)}} D\right)^{d_1} \left(\delta_2^{\frac{d_2+1}{d_2(d_2-1)}} D\right)^{d_2} D^3\right) && (\text{by definition of } \delta_1 \text{ and } \delta_2) \\
&= \Omega\left(D^{\frac{d_1+1}{d_1-1} + d_1 + \frac{d_2+1}{d_2-1} + d_2 + 3}\right) && (\delta_1 \geq \delta_2 \geq D)
\end{aligned}$$

The exponent is minimized when d_1 and d_2 are either 2 or 3. By plugging in $d_1 = d_2 = 2$, we get $n = \Omega(D^{13})$ and hence the additive stretch $D = O(n^{1/13})$. This admits a construction with parameters $d_1 = d_2 = 2$, $D = \Theta(n^{1/13})$, $r = \Theta(n^{3/26})$ and $R = \Theta(n^{5/26})$. ◀

▶ **Corollary 13.** *Fix an $\epsilon \in [0, 1/3)$ and let d be such that $\epsilon \in \left[0, \frac{d-1}{3d+1}\right]$. There exists a graph G with n vertices such that any spanner $H \subseteq G$ with $O(n^{1+\epsilon})$ edges has additive stretch $+\Omega\left(n^{\left(1-\frac{3d+1}{d-1}\epsilon\right)/\left(3+2d+2\frac{d+1}{d-1}\right)}\right)$. In particular, by setting $d = 3$ the additive stretch becomes $\Omega(n^{\frac{1}{13}-\frac{5}{13}\epsilon})$.*

3.2 $O(n)$-sized Emulators

▶ **Definition 14.** Let $G = (V, E)$ be an (unweighted) undirected graph. A weighted graph $H = (V, E', w)$ is said to be an *emulator with additive stretch β* if for any two vertices $x, y \in V$, $\text{dist}_G(x, y) \leq \text{dist}_H(x, y) \leq \text{dist}_G(x, y) + \beta$.

The difference between emulators and spanners is that emulators can use weighted edges not present in G. The lower bound graph we use is constructed exactly as in Section 3.1, but with different numerical parameters.

▶ **Theorem 15.** *There exists an undirected graph G with n vertices such that any emulator with $O(n)$ edges has $+\Omega(n^{1/18})$ additive stretch.*

▶ **Lemma 16.** *Every emulator with additive stretch $+(2D-1)$ on G, requires at least $|P|/2$ edges.*

Proof. Let H be an emulator with additive stretch $+(2D-1)$. Without loss of generality, we may assume that any $(u, v) \in E(H)$ has weight precisely $\text{dist}_G(u, v)$. (It is not allowed to be smaller, and it is unwise to make it larger.) We proceed to convert H into a spanner H' that has the same stretch $+(2D-1)$ on all pairs in P, then apply Lemma 12.

Initially H' is empty. Consider each $(x, y) \in P$ one at a time. Let $P_{(x,y)}$ be the shortest path in H and $P'_{(x,y)}$ be the corresponding path in G. Include the entire path $P'_{(x,y)}$ in H'. After this process is complete, for any $(x, y) \in P$, $\text{dist}_{H'}(x, y) = \text{dist}_H(x, y)$, and H' is a spanner with at most $n + 2D|H|$ edges. In particular, it has at most $2D|H|$ clique edges since each weighted edge in some $P_{(x,y)}$ contributes at most $2D$ clique edges to H'. By Lemma 12, the number of clique edges in H' is at least $D|P|$, hence $|H| \geq |P|/2$. ◀

Proof of Theorem 15. In order to get $|P| = \Omega(n)$, it suffices to set $\delta_1 \geq \delta_2 \geq D^2$.

Now, we have

$$\begin{aligned}
n &= \Theta\left(R_1^{d_1} R_2^{d_2} D^2 \delta_1\right) \\
&= \Omega\left((r_1 D)^{d_1} (r_2 D)^{d_2} D^4\right) && (\delta_1 \geq \delta_2 \geq D^2) \\
&= \Omega\left(\left(\delta_1^{\frac{d_1+1}{d_1(d_1-1)}} D\right)^{d_1} \left(\delta_2^{\frac{d_2+1}{d_2(d_2-1)}} D\right)^{d_2} D^4\right) && (\text{by definition of } \delta_1 \text{ and } \delta_2) \\
&= \Omega\left(D^{2\frac{d_1+1}{d_1-1}+d_1+2\frac{d_2+1}{d_2-1}+d_2+4}\right) && (\delta_1 \geq \delta_2 \geq D^2)
\end{aligned}$$

The exponent is minimized when $d_1 = d_2 = 3$. This implies $n = \Omega(D^{18})$. Thus, we have the additive stretch $D = O(n^{1/18})$. ◀

▶ **Corollary 17.** *Fix an* $\epsilon \in [0, 1/3)$ *and let* d *be such that* $\epsilon \in \left[0, \frac{d-1}{3d+1}\right]$. *There exists a graph* G *with* n *vertices such that any emulator* H *with* $O(n^{1+\epsilon})$ *edges has additive stretch* $+\Omega\left(n^{\left(1-\frac{3d+1}{d-1}\epsilon\right)/\left(4+2d+2\frac{d+1}{d-1}\right)}\right)$. *In particular, by setting* $d = 3$ *the additive stretch lowerbound becomes* $\Omega(n^{\frac{1}{18}-\frac{5}{18}\epsilon})$.

Using the same proof technique as in [1, 2], it is possible to extend our emulator lower bound to *any* compressed representation of graphs using $\tilde{O}(n)$ bits.

▶ **Theorem 18.** *Consider any mapping from* n*-vertex graphs to* $\tilde{O}(n)$*-length bitstrings. Any algorithm for reconstructing an approximation of* dist_G*, given the bitstring encoding of* G*, must have additive error* $+\tilde{\Omega}(n^{1/18})$.

Proof. For each subset $T \subseteq P$ construct the graph G_T by removing all clique edges from G that are on the critical paths of pairs in T. Because all clique edges are missing, for all $(x, y) \in T$ we have $d_{G_T}(x, y) \geq d_G(x, y) + 2D$. On the other hand, for all $(x, y) \notin T$, $d_{G_T}(x, y) = d_G(x, y)$.

There are $2^{|P|}$ such graphs. If we represent all such graphs with bitstrings of length $|P| - 1$ then by the pigeonhole principle two such graphs G_T and $G_{T'}$ are mapped to the same bitstring. Let (x, y) be any pair in $T \backslash T'$. Since $\text{dist}_{G_T}(x, y) \geq \text{dist}_{G_{T'}}(x, y) + 2D$, the additive stretch of any such scheme must be at least $2D$. Alternatively, any scheme with stretch $2D - 1$ must use bitstrings of length at least length $|P|$.

Now, by setting $d = 3$ with $D = \tilde{\Theta}(n^{1/18})$, $r_1 = r_2 = \tilde{\Theta}(n^{2/27})$ and $R_1 = R_2 = \tilde{\Theta}(n^{7/54})$, we have $|P| = \tilde{\Theta}(n)$. Thus any $\tilde{O}(n)$-length encoding must recover approximate distances with stretch $+\tilde{\Omega}(n^{1/18})$. ◀

4 Conclusion

Our constructions, like [1, 12, 2, 16], are based on looking at the convex hulls of integer lattice points in $\mathbb{Z}^d$ lying in a ball of some radius. Whereas Theorems 15 and 18 hold for $d = 3$, Theorems 1, 8, and 10 are indifferent between dimensions $d = 2$ and $d = 3$, but that is only because d *must be an integer*.

Suppose we engage in a little magical thinking, and imagine that there are integer lattices in any *fractional* dimension, and moreover, that some analogue of Bárány and Larman's [10] bound holds in these lattices. If such objects existed then we could obtain slightly better lower bounds. For example, setting $d = 1 + \sqrt{2}$ in the proof of Theorem 1, we would conclude that any $O(n)$-size shortcut set cannot reduce the diameter below $\Omega(n^{1/(3+2\sqrt{2})})$, which is an improvement over $\Omega(n^{1/6})$ as $3 + 2\sqrt{2} < 5.83$.

For near-linear size spanners and emulators there are still large gaps between the best lower and upper bounds on additive stretch: $[n^{1/13}, n^{3/7}]$ in the case of spanners and $[n^{1/18}, n^{1/4}]$ in the case of emulators. None of the existing lower or upper bound techniques seem up to the task of closing these gaps entirely.

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, pages 28:1–28:20, 2017.

2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2017.

3 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

4 Noga Alon. Testing subgraphs in large graphs. *Random Structures & Algorithms*, 21(3-4):359–370, 2002.

5 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (α, β)-spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010.

6 Felix Behrend. On sets of integers which contain no three terms in arithmetic progression. *Proc. Nat. Acad. Sci.*, 32:331–332, 1946.

7 Greg Bodwin. Linear size distance preservers. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 600–615, 2017.

8 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016.

9 Gregory Bodwin and Virginia Vassilevska Williams. Very sparse additive spanners and emulators. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 377–382, 2015.

10 Imre Bárány and David G. Larman. The convex hull of the integer points in a large ball. *Mathematische Annalen*, 312(1):167–181, 1998.

11 Shiri Chechik. New additive spanners. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 498–512, 2013.

12 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, pages 463–501, 2006.

13 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

14 Michael Elkin and David Peleg. $(1 + \epsilon,\beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

15 Jeremy T. Fineman. Nearly work-efficient parallel algorithm for digraph reachability. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, 2018.

16 William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

17 Shang-En Huang and Seth Pettie. Thorup-Zwick emulators are universally optimal hopsets. *CoRR*, abs/1705.00327, 2017. `arXiv:1705.00327`.

18 Mathias Bæk Tejs Knudsen. Additive spanners: A simple construction. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 277–281, 2014.

19 Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1):7, 2009.

20 Mikkel Thorup. On shortcutting digraphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 205–211. Springer, 1992.

21 Mikkel Thorup. Shortcutting planar digraphs. *Combinatorics, Probability and Computing*, 4(3):287–315, 1995.

22 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

23 David P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

Reconfiguration of Colorable Sets in Classes of Perfect Graphs

Takehiro Ito

Graduate School of Information Sciences, Tohoku University.
Aoba-yama 6-6-05, Sendai, 980-8579, Japan
takehiro@ecei.tohoku.ac.jp
https://orcid.org/0000-0002-9912-6898

Yota Otachi

Faculty of Advanced Science and Technology, Kumamoto University.
2-39-1 Kurokami, Chuo-ku, Kumamoto, 860-8555 Japan
otachi@cs.kumamoto-u.ac.jp
https://orcid.org/0000-0002-0087-853X

Abstract

A set of vertices in a graph is *c-colorable* if the subgraph induced by the set has a proper c-coloring. In this paper, we study the problem of finding a step-by-step transformation (reconfiguration) between two c-colorable sets in the same graph. This problem generalizes the well-studied Independent Set Reconfiguration problem. As the first step toward a systematic understanding of the complexity of this general problem, we study the problem on classes of perfect graphs. We first focus on interval graphs and give a combinatorial characterization of the distance between two c-colorable sets. This gives a linear-time algorithm for finding an actual shortest reconfiguration sequence for interval graphs. Since interval graphs are exactly the graphs that are simultaneously chordal and co-comparability, we then complement the positive result by showing that even deciding reachability is PSPACE-complete for chordal graphs and for co-comparability graphs. The hardness for chordal graphs holds even for split graphs. We also consider the case where c is a fixed constant and show that in such a case the reachability problem is polynomial-time solvable for split graphs but still PSPACE-complete for co-comparability graphs. The complexity of this case for chordal graphs remains unsettled. As by-products, our positive results give the first polynomial-time solvable cases (split graphs and interval graphs) for Feedback Vertex Set Reconfiguration.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases reconfiguration, colorable set, perfect graph

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.27

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.06511`.

Funding T.I. was partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Number JP16K00004, Japan. Y.O. was partially supported by MEXT KAKENHI Grant Number JP24106004, JSPS KAKENHI Grant Number JP25730003, and by FY 2015 Researcher Exchange Program between JSPS and NSERC.

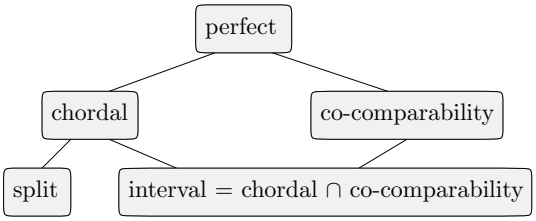


Figure 1 The graph classes studied in this paper.

1 Introduction

Recently, the reconfiguration framework has been applied to several search problems. In a reconfiguration problem, we are given two feasible solutions of a search problem and are asked to determine whether we can modify one to the other by repeatedly applying prescribed reconfiguration rules while keeping the feasibility (see [14, 22, 19]). Studying such a problem is important for understanding the structure of the solution space of the underlying problem. Computational complexity of reconfiguration problems has been studied intensively. For example, the Independent Set Reconfiguration problem under the reconfiguration rules TS [13], TAR [14], and TJ [15] has been studied for several graph classes such as planar graphs [13], perfect graphs [15], claw-free graphs [5], trees [6], interval graphs [4], and bipartite graphs [17].

In this paper, we initiate the study on the problem of reconfiguring *colorable sets*, which generalizes Independent Set Reconfiguration. For a graph $G = (V, E)$ and an integer $c \geq 1$, a vertex set $S \subseteq V$ is *c-colorable* if the subgraph $G[S]$ induced by S admits a proper c-coloring. For example, the 1-colorable sets in a graph are exactly the independent sets of the graph. Recently, c-colorable sets have been studied from the viewpoint of wireless network optimization (see [2, 3] and the references therein). The Colorable Set Reconfiguration problem asks given two c-colorable sets S and S' in a graph G, whether we can reach from S to S' by repeatedly applying allowed local changes. We consider the following three local change operations (see Section 2 for formal definitions):

- TAR(k): either adding or removing one vertex while keeping the size of the set at least a given threshold k.
- TJ: swap one member for one nonmember.
- TS: swap one member for one nonmember adjacent to the member.

In perfect graphs, being c-colorable is equivalent to having no clique of size more than c. This property often makes problems related to coloring tractable. Thus, to understand this very general problem, we start the study of Colorable Set Reconfiguration on classes of perfect graphs. Figure 1 shows the graph classes studied in this paper and the inclusion relationships (see Section 2.2 for definitions).

Our contribution

Before we start our investigation on the reconfiguration problem, we first fill a gap in the complexity landscape of the search problem Colorable Set that asks for finding a large c-colorable set. When $c = 1$, Colorable Set is equivalent to the classical problem of finding a large independent set that can be solved in polynomial time for perfect graphs. For larger c, it was only known that the case $c = 2$ is NP-complete for perfect graphs [1]. To

Table 1 Summary of the results. PSPACE-completeness results here apply to TS also, while polynomial-time algorithms do not. The case of $c = 1$ is equivalent to Independent Set Reconfiguration.

	Colorable Set Reconfiguration under TAR/TJ		
	$c = 1$	fixed $c \geq 2$	arbitrary c
perfect	PSPACE-c		
co-comparability	PSPACE-c [15][2]	PSPACE-c (omitted)	
chordal	P [15]	?	PSPACE-c
split	P	P (Thm 4.4)	PSPACE-c (Thm 4.5)
interval	P	P (Thm 3.11)	
bipartite	NP-c [17]	Trivial if $c \geq 2$	

make the complexity status of Colorable Set for perfect graphs complete, we show that it is NP-complete for any fixed $c \geq 2$.[1]

We then show complexity divergences among the classes of perfect graphs in Figure 1, in particular under TAR and TJ. See Table 1 for a summary of our results. Our results basically say that the problem under TAR and TJ is tractable on interval graphs but further generalization is not quite possible.

More specifically, we first study the problem on interval graphs and show that a shortest reconfiguration sequence under TAR can be found in linear time (Theorem 3.11). This implies the same result under TJ. Next we study the problem on split graphs. We show that the complexity depends on c. When c is a fixed constant, the problem is polynomial-time solvable under TAR and TJ (Theorem 4.4). If c is a part of input, then we can show that the problem is PSPACE-complete under all rules, including TS (Theorem 4.5). While the hardness result applies also to chordal graphs, it is unclear whether a similar positive result for chordal graphs can be obtained when c is a fixed constant. We only know that the case of $c = 1$ under TAR and TJ is polynomial-time solvable as chordal graphs are even-hole-free [15]. We also show that for every fixed $c \geq 1$ the problem is PSPACE-complete for co-comparability graphs under all rules.[3] Thus, our results are in some sense tight since the interval graphs are exactly the chordal co-comparability graphs and split graphs are chordal graphs (see Figure 1).

As a byproduct of Theorems 3.11 and 4.4, the Feedback Vertex Set Reconfiguration problem [18] turns out to be polynomial-time solvable for split graphs and interval graphs under TAR and TJ. These are the first polynomial-time solvable cases for Feedback Vertex Set Reconfiguration. To see the polynomial-time solvability, observe that the complements $V(G) \setminus S$ of 2-colorable sets S in a chordal graph G are exactly the feedback vertex sets in the graph[4] and reconfigurations of the complements are equivalent to reconfigurations of the original vertex sets under TAR and TJ.

[1] This result is omitted in this version.
[2] The reduction in [15] outputs co-comparability graphs.
[3] This result is omitted in this version.
[4] Each induced cycle in a chordal graph is a triangle, and thus 2-colorable (or equivalently, odd cycle free) chordal graphs are forests.

2 Preliminaries

We say, as usual, that an algorithm for a graph $G = (V, E)$ runs in *linear time* if the running time of the algorithm is $O(|V| + |E|)$.

A *proper c-coloring* of a graph assigns a color from $\{1, \ldots, c\}$ to each vertex in such a way that adjacent vertices have different colors. Given a graph G and an integer c, Graph Coloring asks whether G admits a proper c-coloring. This problem is NP-complete even if c is fixed to 3 [9]. The minimum c such that a graph admits a proper c-coloring is its *chromatic number*.

The Colorable Set problem is a generalization of Graph Coloring where we find a large induced subgraph of the input graph that admits a proper c-coloring. Let $G = (V, E)$ be a graph. For a set of vertices $S \subseteq V$, we denote by $G[S]$ the subgraph induced by S. A vertex set $S \subseteq V$ is *c-colorable* in G if $G[S]$ has a proper c-coloring. Now the problem is defined as follows:

Problem: Colorable Set
Input: A graph G and integers c and k.
Question: Does G have a c-colorable set of size at least k?

The problem of finding a large c-colorable set is studied for a few important classes of perfect graphs (see Figure 1 and Table 1). For the class of perfect graphs, it is known that a maximum 1-colorable set (that is, a maximum independent set) can be found in polynomial time [12]. Parameterized complexity [16] and approximation [7] of Colorable Set on perfect graphs are also studied.

2.1 Reconfiguration of colorable sets

Let S and S' be c-colorable sets in a graph G. Then, $S \leftrightarrow S'$ *under* $\mathsf{TAR}(k)$ for a nonnegative integer k if $|S|, |S'| \geq k$ and $|S \triangle S'| = 1$, where $S \triangle S'$ denotes the symmetric difference $(S \setminus S') \cup (S' \setminus S)$. Here $S \leftrightarrow S'$ means that S and S' can be reconfigured to each other in one step and TAR stands for "token addition & removal." A sequence $\langle S_0, S_1, \ldots, S_\ell \rangle$ of c-colorable sets in G is a *reconfiguration sequence* of length ℓ between S_0 and S_ℓ under $\mathsf{TAR}(k)$ if $S_{i-1} \leftrightarrow S_i$ holds under $\mathsf{TAR}(k)$ for all $i \in \{1, 2, \ldots, \ell\}$. A reconfiguration sequence under $\mathsf{TAR}(k)$ is simply called a *$\mathsf{TAR}(k)$-sequence*. We write $S_0 \leftrightsquigarrow S_\ell$ *under* $\mathsf{TAR}(k)$ if there exists a $\mathsf{TAR}(k)$-sequence between S_0 and S_ℓ. Note that every reconfiguration sequence is *reversible*, that is, $S_0 \leftrightsquigarrow S_\ell$ if and only if $S_\ell \leftrightsquigarrow S_0$. Now the problem we are going to consider is formalized as follows:

Problem: Colorable Set Reconfiguration under TAR ($\mathrm{CSR}_{\mathsf{TAR}}$ for short)
Input: A graph G, integers c and k, and c-colorable sets S and S' of G.
Question: Does $S \leftrightsquigarrow S'$ under $\mathsf{TAR}(k)$ hold?

We denote by (G, c, S, S', k) an instance of $\mathrm{CSR}_{\mathsf{TAR}}$. We assume that both $|S| \geq k$ and $|S'| \geq k$ hold; otherwise it is trivially a no-instance. Note that the lower bound k guarantees that none of the sets in the reconfiguration sequence is too small. Without the lower bound, the reachability problem becomes trivial as S can always reach S' via $\emptyset$.

For a $\mathrm{CSR}_{\mathsf{TAR}}$-instance (G, c, S, S', k), we denote by $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$ the length of a shortest $\mathsf{TAR}(k)$-sequence in G between S and S'; if there is no such a sequence, then we set $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = \infty$.

We note that $\mathrm{CSR}_{\mathsf{TAR}}$ is a decision problem and hence does not require the specification of an actual $\mathsf{TAR}(k)$-sequence. Similarly, the shortest variant of $\mathrm{CSR}_{\mathsf{TAR}}$ simply requires to output the value of $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$.

Other reconfiguration rules

Although the TAR rule is our main target, we also study two other well-known rules TJ (token jumping) and TS (token sliding). Let S and S' be c-colorable sets in a graph G. For TJ and TS, we additionally assume that $|S| = |S'|$ because these rules do not change the size of a set. Now the rules are defined as follows:

- $S \leftrightarrow S'$ under TJ if $|S \setminus S'| = |S' \setminus S| = 1$;
- $S \leftrightarrow S'$ under TS if $|S \setminus S'| = |S' \setminus S| = 1$ and the two vertices in $S \triangle S'$ are adjacent in G.

Reconfiguration sequences under TJ and TS as well as the reconfiguration problems $\mathrm{CSR}_{\mathsf{TJ}}$ and $\mathrm{CSR}_{\mathsf{TS}}$ are defined analogously. An instance of $\mathrm{CSR}_{\mathsf{TJ}}$ or $\mathrm{CSR}_{\mathsf{TS}}$ is represented as (G, c, S, S'), and $\mathsf{dist}_{\mathsf{TJ}}(S, S')$ and $\mathsf{dist}_{\mathsf{TS}}(S, S')$ are defined in the same way.

The following relation can be shown in almost the same way as Theorem 1 in [15] and means that $\mathrm{CSR}_{\mathsf{TJ}}$ is not harder than $\mathrm{CSR}_{\mathsf{TAR}}$ in the sense of Karp reductions.

▶ **Lemma 2.1.** *Let S and S' be c-colorable sets of size $k+1$ in a graph G. Then, $S \leftrightsquigarrow S'$ under $\mathsf{TAR}(k)$ if and only if $S \leftrightsquigarrow S'$ under TJ. Furthermore, it holds that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = 2 \cdot \mathsf{dist}_{\mathsf{TJ}}(S, S')$.*

To make the presentation easier, we often use the shorthands $S + v$ for $S \cup \{v\}$ and $S - v$ for $S \setminus \{v\}$. For a vertex v of a graph G, we denote the neighborhood of v in G by $N_G(v)$.

2.2 Graph classes

A *clique* in a graph is a set of pairwise adjacent vertices. A graph is *perfect* if the chromatic number equals the maximum clique size for every induced subgraph [11]. The following fact follows directly from the definition of perfect graphs and will be used throughout this paper.

▶ **Observation 2.2.** *A vertex set $S \subseteq V(G)$ of a perfect graph G is c-colorable if and only if $G[S]$ has no clique of size more than c.*

There are many subclasses of perfect graphs. Chordal graphs form one of the most well-known subclasses of perfect graphs, where a graph is *chordal* if it contains no induced cycle of length greater than 3.

Co-comparability graphs form another large class of perfect graphs. A graph $G = (V, E)$ is a *co-comparability graph* if there is a linear ordering $\prec$ on V such that $u \prec v \prec w$ and $\{u, w\} \in E$ imply $\{u, v\} \in E$ or $\{v, w\} \in E$. Although they are less known than chordal graphs, co-comparability graphs generalize several important graph classes such as interval graphs, permutation graphs, trapezoid graphs, and co-bipartite graphs (see [11, 20]).

The classes of chordal graphs and co-comparability graphs are incomparable.[5] It is known that the class of interval graphs characterizes their intersection; namely, a graph is an interval graph if and only if it is a co-comparability graph and chordal [10]. Recall that a graph is an *interval graph* if it is the intersection graph of closed intervals on the real line.

Another well-studied subclass of chordal graphs (and hence of perfect graphs) is the class of split graphs. A graph $G = (V, E)$ is a *split graph* if V can be partitioned into a clique K and an independent set I. To emphasize that G is a split graph, we write $G = (K, I; E)$. The classes of interval graphs and split graphs are incomparable.[6]

[5] A cycle of four vertices is a co-comparability graph but not chordal. The *net* graph obtained by attaching a pendant vertex to each vertex of a triangle is chordal but not a co-comparability graph.

[6] A path with five or more vertices is an interval graph but not a split graph. The net graph is a split graph but not an interval graph.

3 Shortest reconfiguration in interval graphs

In this section, we show that $\mathrm{CSR}_{\mathsf{TAR}}$ for interval graphs can be solved in linear time. Our result is actually stronger and says that an actual shortest $\mathsf{TAR}(k)$-sequence can be found in linear time, if one exists. By Lemma 2.1, the same result is obtained for TJ. We first give a characterization of the distance between two c-colorable sets in an interval graph (Section 3.1). This characterization says that a shortest $\mathsf{TAR}(k)$-sequence has length linear in the number of vertices of the graph. We then show that the distance can be computed in linear time (Section 3.2). We finally present a linear-time algorithm for finding a shortest $\mathsf{TAR}(k)$-sequence (Section 3.3).

It is known that a graph is an interval graph if and only if its maximal cliques can be ordered so that each vertex appears consecutively in that ordering [10, 8]. We call a list of the maximal cliques ordered in such a way a *clique path*. Let $G = (V, E)$ be an interval graph and $(M_1, \ldots, M_t)$ be a clique path of G; that is, for each vertex $v \in V$, there are indices l_v and r_v such that $v \in M_i$ if and only if $l_v \leq i \leq r_v$. Given an interval graph, a clique path and the indices l_v and r_v for all vertices can be computed in linear time [21]. Hence we can assume that we are additionally given such information. Note that $\mathcal{I} = \{[l_v, r_v] : v \in V\}$ is an interval representation of G. Namely, $\{u, v\} \in E$ if and only $[l_u, r_u] \cap [l_v, r_v] \neq \emptyset$.

Let K be a clique in an interval graph G. By the Helly property of intervals, the intersection of all intervals in K is nonempty; that is, $\bigcap_{v \in K} [l_v, r_v] \neq \emptyset$ (see [20]). A point in the intersection $\bigcap_{v \in K} [l_v, r_v]$ is a *clique point* of K.

3.1 The distance between c-colorable sets

Let (G, c, S, S', k) be an instance of $\mathrm{CSR}_{\mathsf{TAR}}$. The set S is *locked in* G if S is a maximal c-colorable set in G and $|S| = k$. The following lemma follows immediately from the definition.

▶ **Lemma 3.1.** *Let G be a graph, and let S and S' be distinct c-colorable sets of size at least k in G. If S or S' is locked in G, then $S \not\leftrightsquigarrow S'$.*

Proof. Assume without loss of generality that S is locked in G. If there is a c-colorable set S_1 in G such that $S \leftrightarrow S_1$, then $S \subsetneq S_1$ as $|S| = k$. This contradicts the maximality of S. Since $S \neq S'$, we can conclude that $S \not\leftrightsquigarrow S'$. ◀

The rest of this subsection is dedicated to a proof of the following theorem, which implies that the converse of the lemma above also holds for interval graphs.

▶ **Theorem 3.2.** *Let G be an interval graph, and let S and S' be distinct c-colorable sets of size at least k in G. If S and S' are not locked in G, then the distance $d := \mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$ is determined as follows.*

1. *If S and S' are not locked in $G[S \cup S']$, then $d = |S \triangle S'|$.*
2. *If exactly one of S and S' is locked in $G[S \cup S']$, then $d = |S \triangle S'| + 2$.*
3. *If S and S' are locked in $G[S \cup S']$, then we have the following two cases.*
   a. *If there is $v \in V(G) \setminus (S \cup S')$ such that both $S + v$ and $S' + v$ are c-colorable in G, then $d = |S \triangle S'| + 2$.*
   b. *Otherwise, $d = |S \triangle S'| + 4$.*

▶ **Corollary 3.3.** *For $S \neq S'$, $S \leftrightsquigarrow S'$ if and only if none of S and S' is locked in G.*

Observe that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \geq |S \triangle S'|$ for any pair of c-colorable sets S and S' in G. We use this fact implicitly in the following arguments.

▶ **Lemma 3.4** (Theorem 3.2 (1)). *Let G be an interval graph, and let S and S' be c-colorable sets of size at least k in G. If S and S' are not locked in $G[S \cup S']$, then* $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = |S \vartriangle S'|$.

Proof. We proceed by induction on $|S \vartriangle S'|$. The base case of $|S \vartriangle S'| = 0$ is trivial. Assume that $|S \vartriangle S'| > 0$ and that the statement is true if the symmetric difference is smaller.

We first consider the case where $|S| = k$. Since S is not locked in $G[S \cup S']$, S is not maximal in $G[S \cup S']$. Thus there is a vertex $v \in S' \setminus S$ such that $T := S + v$ is c-colorable. The set T is not locked in $G[T \cup S']$, $S \leftrightarrow T$, and $|T \vartriangle S'| = |S \vartriangle S'| - 1$. By the induction hypothesis, $\mathsf{dist}_{\mathsf{TAR}(k)}(T, S') = |T \vartriangle S'| = |S \vartriangle S'| - 1$. Hence, we have $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \leq \mathsf{dist}_{\mathsf{TAR}(k)}(T, S') + 1 = |S \vartriangle S'|$. If $|S'| = k$, we can apply the same argument.

In the following, we assume that $|S| > k$ and $|S'| > k$. If $S \subseteq S'$, then we can add the elements of $S' \setminus S$ one-by-one in an arbitrary order to get a shortest reconfiguration sequence of length $|S' \setminus S| = |S \vartriangle S'|$. The case where $S' \subseteq S$ is the same.

We now consider the case where $S \not\subseteq S'$ and $S' \not\subseteq S$. Let $v \in S \setminus S'$ and $w \in S' \setminus S$ be vertices with the smallest right-end in each set. That is, $r_v = \min\{r_x : x \in S \setminus S'\}$ and $r_w = \min\{r_x : x \in S' \setminus S\}$. By symmetry, assume that $r_w \leq r_v$. Let $u \in S \setminus S'$ be a vertex that minimizes l_u. (Note that u and v may be the same.) Now we have $r_w \leq r_v \leq r_u$. We set $T = S - u$ and $T' = S - u + w$. Clearly, $S \leftrightarrow T$. To apply the induction hypothesis, it suffices to show that T is not locked in $G[T \cup S']$. To this end, we prove that $T \leftrightarrow T'$. Suppose to the contrary that T' is not c-colorable; that is, T' contains a clique K of size $c + 1$. Since T does not contain such a large clique, K must include w. Let p be a clique point of K. If $p < l_u$, then K includes no vertex in $S \setminus S'$ as u has the minimum l_u in $S \setminus S'$. This contradicts the c-colorability of S' and thus $l_u \leq p \leq r_w \leq r_u$. This implies that $K - w + u \subseteq S$ is a clique of size $c + 1$, a contradiction. Therefore, we can conclude that T' is c-colorable. Now, by the induction hypothesis, $\mathsf{dist}_{\mathsf{TAR}(k)}(T, S') = |T \vartriangle S'| = |S \vartriangle S'| - 1$, and thus $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \leq \mathsf{dist}_{\mathsf{TAR}(k)}(T, S') + 1 = |S \vartriangle S'|$. ◀

▶ **Lemma 3.5** (Theorem 3.2 (2)). *Let G be an interval graph, and let S and S' be distinct c-colorable sets of size at least k in G. If S and S' are not locked in G, and exactly one of S and S' is locked in $G[S \cup S']$, then* $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = |S \vartriangle S'| + 2$.

Proof. Without loss of generality, assume that S is locked in $G[S \cup S']$. This implies that $|S| = k$. Since S is not locked in G, S is not maximal in G. Hence, there is a vertex $v \in V(G) \setminus (S \cup S')$ such that $T := S + v$ is a c-colorable set of G. Observe that T and S' are not locked in $G[T \cup S']$. Thus, by Theorem 3.2 (1), it holds that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \leq \mathsf{dist}_{\mathsf{TAR}(k)}(T, S') + 1 = |T \vartriangle S'| + 1 = |S \vartriangle S'| + 2$.

On the other hand, since S is locked in $G[S \cup S']$, every c-colorable set T of G with $S \leftrightarrow T$ contains a vertex in $V(G) \setminus (S \cup S')$. Thus $|T \vartriangle S'| = |S \vartriangle S'| + 1$ holds. This implies that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \geq \min_{T\colon S \leftrightarrow T} |T \vartriangle S'| + 1 = |S \vartriangle S'| + 2$. ◀

▶ **Lemma 3.6** (Theorem 3.2 (3a)). *Let G be an interval graph, and let S and S' be distinct c-colorable sets of size at least k in G. Assume S and S' are locked in $G[S \cup S']$ but not in G. If there is a vertex $v \in V(G) \setminus (S \cup S')$ such that both $S + v$ and $S' + v$ are c-colorable in G, then* $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = |S \vartriangle S'| + 2$.

Proof. Let $v \in V(G) \setminus (S \cup S')$ be a vertex such that both $S + v$ and $S' + v$ are c-colorable in G. We have $S \leftrightarrow S + v$ and $S' \leftrightarrow S' + v$. Since $S + v$ and $S' + v$ are not locked in $G[S \cup S' + v]$, Theorem 3.2 (1) implies that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \leq \mathsf{dist}_{\mathsf{TAR}(k)}(S + v, S' + v) + 2 = |(S + v) \vartriangle (S' + v)| + 2 = |S \vartriangle S'| + 2$.

The lower bound $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \geq |S \triangle S'| + 2$ can be shown in exactly the same way as in the proof of Lemma 3.5. ◀

▶ **Lemma 3.7** (Theorem 3.2 (3b)). *Let G be an interval graph, and let S and S' be distinct c-colorable sets of size at least k in G. Assume S and S' are locked in $G[S \cup S']$ but not in G. If there is no vertex $v \in V(G) \setminus (S \cup S')$ such that both $S + v$ and $S' + v$ are c-colorable in G, then $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') = |S \triangle S'| + 4$.*

Proof. Let $u, v \in V(G) \setminus (S \cup S')$ be distinct vertices such that $S+u$ and $S'+v$ are c-colorable in G. Since $S + u$ and $S' + v$ are not locked in $G[(S + u) \cup (S' + v)]$, Theorem 3.2 (1) implies that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \leq \mathsf{dist}_{\mathsf{TAR}(k)}(S + u, S' + v) + 2 = |(S + u) \triangle (S' + v)| + 2 = |S \triangle S'| + 4$.

Since S and S' are locked in $G[S \cup S']$ and there is no vertex $v \in V(G) \setminus (S \cup S')$ such that both $S + v$ and $S' + v$ are c-colorable in G, we have $\min_{T\colon S \leftrightarrow T,\ T'\colon S' \leftrightarrow T'} |T \triangle T'| = |S \triangle S'| + 2$. This implies that $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S') \geq |S \triangle S'| + 4$. ◀

3.2 Computing the distance in linear time

We here explain how to check which case of Theorem 3.2 applies to a given instance in linear time.

▶ **Lemma 3.8.** *Given an interval graph G and c-colorable sets S and S' in G, one can either find a vertex $v \notin S \cup S'$ such that $S + v$ and $S' + v$ are c-colorable or decide that no such vertex exists in linear time.*

Proof. Let $(M_1, \ldots, M_t)$ be a clique path of G. Recall that $M_1, \ldots, M_t$ are the maximal cliques of G. Thus, for every $T \subseteq V(G)$, the maximum clique size of $G[T]$ is equal to $\max_{1 \leq i \leq t} |T \cap M_i|$.

We compute $a_i^S = |S \cap M_i|$ for $1 \leq i \leq t$ as follows. Initialize all a_i^S to 0; for each $u \in S$, add 1 to all a_i^S with $l_u \leq i \leq r_u$. In the same way, we compute $a_i^{S'} = |S' \cap M_i|$ for $1 \leq i \leq t$. From the observation above, we can conclude that for each vertex $v \notin S \cup S'$, $S + v$ and $S' + v$ are c-colorable if and only if $a_i^S, a_i^{S'} < c$ for $l_v \leq i \leq r_v$.

The initialization and the test for all nonmembers of S can be done in time $O(\sum_{i=1}^{t} |M_i|)$. It suffices to show that $\sum_{i=1}^{t} |M_i| \leq \sum_{v \in V(G)} (\deg(v) + 1) = 2|E(G)| + |V(G)|$. Since $M_1 \not\subseteq M_2$, there is a vertex v with $l_v = r_v = 1$. Thus $|M_1| = \deg(v) + 1$. By induction on the number of vertices, our claim holds. ◀

By setting $S = S'$ in the lemma above, we have the following lemma.

▶ **Lemma 3.9.** *Given an interval graph G and a c-colorable set S in G, one can either find a vertex $v \notin S$ such that $S + v$ is c-colorable or decide that S is maximal in linear time.*

▶ **Corollary 3.10.** *Given an interval graph G and c-colorable sets S and S' in G, the distance $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$ can be computed in linear time.*

Proof. We first check whether S or S' is locked in G. If so, the distance is ∞. Otherwise, we check whether S and S' are locked in $G[S \cup S']$. If not both of them are locked in $G[S \cup S']$, then we can apply Theorem 3.2 (1) or (2) and determine the distance. If both S and S' are locked in $G[S \cup S']$, we find a vertex $v \notin S \cup S'$ such that both $S + v$ and $S' + v$ are c-colorable in G. Everything can be done in linear time by Lemmas 3.8 and 3.9. ◀

3.3 Finding a shortest reconfiguration sequence in linear time

Here we describe how we find an actual shortest reconfiguration sequence in linear time. To this end, we need to be careful about the representation of a reconfiguration sequence. If we always output the whole set, the total running time cannot be smaller than $k \cdot \mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$. However, this product can be quadratic. To avoid this blow up, we output only the difference from the previous set. That is, if the current set is S and the next set is $S + v$ ($S - v$), we output $+v$ ($-v$, resp.). We also fully use the reversible property of reconfiguration sequences and output them sometimes from left to right and sometimes from right to left. For example, we may output a reconfiguration sequence $\langle S_0, \ldots, S_5 \rangle$ as first $S_0 \leftrightarrow S_1 \leftrightarrow S_2$, next $S_5 \leftrightarrow S_4 \leftrightarrow S_3$, then $S_2 \leftrightarrow S_3$. It is straightforward to output the sequence from left to right by using a linear-size buffer.

▶ **Theorem 3.11.** *Given an interval graph G and c-colorable sets S and S' in G, a $\mathsf{TAR}(k)$-sequence of length $\mathsf{dist}_{\mathsf{TAR}(k)}(S, S')$ can be computed in linear time.*

Proof. We first test which case of Theorem 3.2 applies to the given instance. This can be done in linear time as shown in the proof of Corollary 3.10. We reduce Cases (2) and (3) to Case (1). The reductions below can be done in linear time by using Lemmas 3.8 and 3.9.

Assume first that Case (2) applies; that is, S is locked but S' is not in $G[S \cup S']$. We find a vertex $v \in V(G) \setminus (S \cup S')$ such that $S + v$ is a c-colorable set of G. We then add v to S. As we saw in the proof of Lemma 3.5, this is a valid step in a shortest reconfiguration sequence. Furthermore, after this step, S and S' are not locked in $G[S \cup S']$.

Next assume that Case (3) applies; that is, both S and S' are locked in $G[S \cup S']$. We find vertices $u, v \notin S \cup S'$ such that $S + u$ and $S' + v$ are c-colorable in G. In Case (3a), we further ask that $u = v$. We then add u to S and v to S'. The proofs of Lemmas 3.6 and 3.7 imply that these are valid steps in a shortest reconfiguration sequence, and that S and S' are no longer locked in $G[S \cup S']$ after these steps.

We now handle Case (1), where S and S' are not locked in $G[S \cup S']$. Assume that $S \not\subseteq S'$ and $S' \not\subseteq S$ since otherwise finding a shortest sequence is trivial. We first compute two orderings of the vertices in $S \cup S'$: nondecreasing orderings of left-ends l_v and of right-ends r_v. Such orderings can be constructed in linear time from a clique path. We maintain information for each vertex v whether $v \in S \setminus S'$, $v \in S' \setminus S$, or $v \notin S \triangle S'$. Using this information, we can also maintain vertices of the smallest left-end and of the smallest right-end in each of $S \setminus S'$ and $S' \setminus S$.

Let $v \in S \setminus S'$ and $w \in S' \setminus S$ be vertices with the smallest right-end in each set. By symmetry, assume that $r_w \leq r_v$. Let $u \in S \setminus S'$ be a vertex that minimizes l_u. As shown in the proof of Lemma 3.4, $S \leftrightarrow (S - u) \leftrightarrow (S - u + w)$ under $\mathsf{TAR}(k)$ and $|S \triangle S'| = |(S - u + w) \triangle S'| + 2$. We output the two steps $S - u$ and $S - u + w$.

We then set $S := S - u + w$ and update the information as $u, w \notin S \triangle S'$ anymore. We also have to maintain the vertices of the smallest left- and right-ends in each $S \setminus S'$ and $S' \setminus S$. Let $w' \in S' \setminus S$ be a vertex with the smallest right-end. The vertex w can be found by sweeping the nondecreasing ordering of the right-ends from the position of w to the right. The vertex $u' \in S \setminus S'$ with the smallest left-end can be found in an analogous way. Although a single update can take super constant steps, it sums up to a linear number of steps in total since it can be seen as a single left-to-right scan of each nondecreasing ordering. Therefore, the total running time is linear. ◀

4 Split graphs

For split graphs, we consider two cases. In the first case, we assume that c is a fixed constant, and show that the problem under TAR (and TJ) can be solved in $O(n^{c+1})$ time. The second case is the general problem having c as a part of input. We show that in this case the problem is PSPACE-complete under all reconfiguration rules.

4.1 Polynomial-time algorithm for fixed c

Let $G = (K, I; E)$ be a split graph, where K is a clique and I is an independent set. For $C \subseteq K$ with $|C| \leq c$, we define T_C as follows:

$$T_C = \begin{cases} C \cup I & \text{if } |C| < c, \\ C \cup I \setminus \{u \in I : C \subseteq N_G(u)\} & \text{if } |C| = c. \end{cases}$$

We can see that T_C is c-colorable for every $C \subseteq K$ with $|C| \leq c$ as follows. Every clique $K' \subseteq T_C$ includes at most $|C| \leq c$ vertices in C and at most one vertex in I. Since a vertex in $T_C \cap I$ has fewer than c neighbors in C, the maximum clique size of $G[T_C]$ is at most c.

▶ **Lemma 4.1.** *If S is a c-colorable set of G with $|S| \geq k$, then $S \leftrightsquigarrow T_{S\cap K}$ under* TAR(k).

Proof. Note that $T_{S\cap K}$ is c-colorable since S is c-colorable and thus $|S \cap K| \leq c$. We now show that $S \subseteq T_{S\cap K}$, which implies $S \leftrightsquigarrow T_{S\cap K}$.

If $|S \cap K| < c$, then $T_{S\cap K} = (S \cap K) \cup I$, and thus $S \subseteq T_{S\cap K}$. If $|S \cap K| = c$, then $S \cap \{u \in I : (S \cap K) \subseteq N(u)\} = \emptyset$ since S is c-colorable. Thus it holds that $S \subseteq (S \cap K) \cup (I \setminus \{u \in I : (S \cap K) \subseteq N(u)\}) = T_{S\cap K}$. ◀

By the reversibility of reconfiguration sequences, we can reduce the problem as follows.

▶ **Corollary 4.2.** *If S and S' are c-colorable sets of G with $|S| \geq k$ and $|S'| \geq k$, then $S \leftrightsquigarrow S'$ under* TAR(k) *if and only if $T_{S\cap K} \leftrightsquigarrow T_{S'\cap K}$ under* TAR(k).

Now we state the crucial lemma for solving the reduced problem.

▶ **Lemma 4.3.** *Let $C \subseteq K$ and $v \in K \setminus C$. If T_C and T_{C+v} are c-colorable sets of size at least k, then $T_C \leftrightsquigarrow T_{C+v}$ under* TAR(k) *if and only if $|T_{C+v}| \geq k + 1$.*

Proof. To prove the if part, assume that $|T_{C+v}| \geq k + 1$. Then, $T_{C+v} \leftrightarrow T_{C+v} - v$. Since $(T_{C+v} - v) \cap K = C$, it holds that $T_{C+v} - v \leftrightsquigarrow T_C$ by Lemma 4.1. Thus we have $T_{C+v} \leftrightsquigarrow T_C$.

To prove the only-if part, assume that $|T_{C+v}| = k$. If $|C + v| = c$, then T_{C+v} is a maximal c-colorable set and no other c-colorable set of size at least k can be reached from T_{C+v}. Assume that $|C + v| < c$, and hence $|C| < c$. Then, $T_{C+v} = (C + v) \cup I$ and $T_C = C \cup I$. Therefore, we have $|T_C| = |T_{C+v} - v| = k - 1$, a contradiction. ◀

Combining the arguments in this subsection, we are now ready to present a polynomial-time algorithm.

▶ **Theorem 4.4.** *Given an n-vertex split graph $G = (K, I; E)$ and c-colorable sets S and S' of size at least k in G, it can be decided whether $S \leftrightsquigarrow S'$ under* TAR(k) *in time $O(n^{c+1})$.*

Proof. We construct a graph $H = (\mathcal{K}, \mathcal{E})$ from $G = (K, I; E)$ as follows:

$$\mathcal{K} = \{C \subseteq K : |C| \leq c \text{ and } |T_C| \geq k\},$$
$$\mathcal{E} = \{\{C, C + v\} : C, C + v \in \mathcal{K} \text{ and } |T_{C+v}| \geq k + 1\}.$$

For each $C \subseteq K$ with $|C| < c$, the size $|T_C| = |C| + |I|$ can be computed in constant time (assuming that we know the size $|I|$ in advance). If $|C| = c$, then we need to compute the size of $\{u \in I : C \subseteq N(u)\}$. This can be done in time $O(n)$ for each C. In H, each $C \in \mathcal{K}$ is adjacent to at most $|C|$ subsets of C: if $|T_C| > k$, then C is adjacent to all $C - v$ with $v \in C$; otherwise, C has no edge to its subsets. This can be computed in time $O(n)$ for each C. In total, the graph H with $O(n^c)$ vertices and $O(n^c)$ edges can be constructed in time $O(n^{c+1})$. For $C, C' \in \mathcal{K}$, one can decide whether H has a C–C' path in time $O(n^c)$.

Let $C := S \cap K$ and $C' := S' \cap K$. Now, by Corollary 4.2, it suffices to show that $T_C \leftrightsquigarrow T_{C'}$ if and only if there is a path between C and C' in H.

Assume that $T_C \leftrightsquigarrow T_{C'}$. Let $\langle S_1 = T_C, S_2, \ldots, S_p = T_{C'} \rangle$ be a reconfiguration sequence from T_C to $T_{C'}$, and let $C_i = S_i \cap K$ for $1 \leq i \leq p$. Observe that $|C_i \triangle C_{i+1}| \leq 1$ for each $1 \leq i < p$. If $C_i \neq C_{i+1}$, Corollary 4.2 and Lemma 4.3 imply that $\{C_i, C_{i+1}\} \in \mathcal{E}$. Since $C_1 = C$ and $C_p = C'$, we can conclude that H has a C–C' path.

Next assume that there is a path between C and C' in H. Let $(C_1 = C, C_2, \ldots, C_q = C')$ be such a path. Lemma 4.3 and the definition of H together imply that $T_{C_i} \leftrightsquigarrow T_{C_{i+1}}$ for each $1 \leq i < q$. Since $C_1 = C$ and $C_q = C'$, we have $T_C \leftrightsquigarrow T_{C'}$. ◀

4.2 PSPACE-completeness when c is a part of input

Here we show the PSPACE-completeness when c is a part of input. (The proof if omitted in this version.)

▶ **Theorem 4.5.** *Given a split graph and c-colorable sets S and S' of size k in the graph, it is PSPACE-complete to decide whether $S \leftrightsquigarrow S'$ under any of* TS, TJ, *and* TAR$(k-1)$.

5 Concluding remarks

We show that Colorable Set Reconfiguration under TAR/TJ is linear-time solvable on interval graphs. Our results give a sharp contrast of the computational complexity with respect to graph classes, while some cases are left unanswered. One of the main unsettled cases is $\mathrm{CSR}_{\mathsf{TAR}}$ with fixed $c > 1$ for chordal graphs (see Table 1). In particular, what is the complexity of $\mathrm{CSR}_{\mathsf{TAR}}$ with $c = 2$ for chordal graphs? This problem is equivalent to the reconfiguration of feedback vertex sets under TAR on chordal graphs. It would be also interesting to study the shortest variant on split graphs with a constant c.

Our positive results for $\mathrm{CSR}_{\mathsf{TAR}}$ on interval graphs and split graphs (Theorems 3.11 and 4.4) do not imply analogous results for $\mathrm{CSR}_{\mathsf{TS}}$. The complexity of $\mathrm{CSR}_{\mathsf{TS}}$ is not settled for these graph classes even with a fixed constant c. It was only recently shown that if $c = 1$, then $\mathrm{CSR}_{\mathsf{TS}}$ can be solved in polynomial time for interval graphs [4]. For $c \geq 2$, $\mathrm{CSR}_{\mathsf{TS}}$ on interval graphs is left unsettled. For split graphs, although co-NP-hardness of a related problem is known [4], $\mathrm{CSR}_{\mathsf{TS}}$ is not solved for all $c \geq 1$.

References

1 Louigi Addario-Berry, W. Sean Kennedy, Andrew D. King, Zhentao Li, and Bruce A. Reed. Finding a maximum-weight induced k-partite subgraph of an i-triangulated graph. *Discrete Applied Mathematics*, 158(7):765–770, 2010. doi:10.1016/j.dam.2008.08.020.

2 Eyjólfur Ingi Ásgeirsson, Magnús M. Halldórsson, and Tigran Tonoyan. Universal framework for wireless scheduling problems. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80

of *LIPIcs*, pages 129:1–129:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.129.

3 Matthias Bentert, René van Bevern, and Rolf Niedermeier. (Wireless) Scheduling, graph classes, and c-colorable subgraphs. *CoRR*, abs/1712.06481, 2017. arXiv:1712.06481.

4 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2017. doi:10.1007/978-3-319-68705-6_10.

5 Paul S. Bonsma, Marcin Kaminski, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2014. doi:10.1007/978-3-319-08404-6_8.

6 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.

7 Samuel Fiorini, R. Krithika, N. S. Narayanaswamy, and Venkatesh Raman. LP approaches to improved approximation for clique transversal in perfect graphs. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 430–442. Springer, 2014. doi:10.1007/978-3-662-44777-2_36.

8 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15(3):835–855, 1965. doi:10.2140/pjm.1965.15.835.

9 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

10 Paul C. Gilmore and Alan J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964. doi:10.4153/CJM-1964-055-5.

11 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. North Holland, second edition, 2004.

12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

13 Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

14 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

15 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

16 R. Krithika and N. S. Narayanaswamy. Parameterized algorithms for (r, l)-partization. *J. Graph Algorithms Appl.*, 17(2):129–146, 2013. doi:10.7155/jgaa.00288.

17 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 185–195. SIAM, 2018. doi:10.1137/1.9781611975031.13.

18 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/s00453-016-0159-2.

19 Naomi Nishimura. Introduction to reconfiguration. *Preprints*, 2017. 2017090055. doi:10.20944/preprints201709.0055.v1.

20 Jeremy P. Spinrad. *Efficient Graph Representations*. Fields Institute monographs. American Mathematical Society, 2003.

21 Ryuhei Uehara and Yushi Uno. On computing longest paths in small graph classes. *Int. J. Found. Comput. Sci.*, 18(5):911–930, 2007. doi:10.1142/S0129054107005054.

22 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

Tight Lower Bounds for List Edge Coloring

Łukasz Kowalik[1]
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl
https://orcid.org/0000-0002-7546-2969

Arkadiusz Socała[2]
Institute of Informatics, University of Warsaw, Poland
arkadiusz.socala@mimuw.edu.pl

Abstract

The fastest algorithms for edge coloring run in time $2^m n^{O(1)}$, where m and n are the number of edges and vertices of the input graph, respectively. For dense graphs, this bound becomes $2^{\Theta(n^2)}$. This is a somewhat unique situation, since most of the studied graph problems admit algorithms running in time $2^{O(n \log n)}$. It is a notorious open problem to either show an algorithm for edge coloring running in time $2^{o(n^2)}$ or to refute it, assuming the Exponential Time Hypothesis (ETH) or other well established assumptions.

We notice that the same question can be asked for list edge coloring, a well-studied generalization of edge coloring where every edge comes with a set (often called a *list*) of allowed colors. Our main result states that list edge coloring for simple graphs does not admit an algorithm running in time $2^{o(n^2)}$, unless ETH fails. Interestingly, the algorithm for edge coloring running in time $2^m n^{O(1)}$ generalizes to the list version without any asymptotic slow-down. Thus, our lower bound is essentially tight. This also means that in order to design an algorithm running in time $2^{o(n^2)}$ for edge coloring, one has to exploit its special features compared to the list version.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness, Theory of computation → Design and analysis of algorithms, Mathematics of computing → Graph coloring

Keywords and phrases list edge coloring, complexity, ETH lower bound

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.28

Acknowledgements We thank an anonymous reviewer for numerous useful remarks.

1 Introduction

An edge coloring of a graph $G = (V, E)$ is a function $c : E \to \mathbb{N}$ which has different values (called colors) on incident edges. This is one of the most basic graph concepts with plethora of results, including classical theorems of Vizing, Shannon and Kőnig. In the decision problem Edge Coloring we are given a simple graph G and an integer k. The question is if G can be edge colored using only k colors. This is an NP-complete problem, as shown by Holyer [9], similarly to many other natural graph decision problems like Clique, Vertex Coloring, Hamiltonicity or Subgraph Isomorphism. However, there is an intriguing difference between our understanding of Edge Coloring and most of the studied graph problems,

[1] The work of Ł. Kowalik is a part of the project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 677651).
[2] Supported by the National Science Centre of Poland, grant number 2015/17/N/ST6/01224.

including the four mentioned above. Namely, the latter ones admit algorithms running in time $2^{O(n \log n)}$, and often even $2^{O(n)}$ for an n-vertex input graph, while it is not known whether Edge Coloring can be solved in time $2^{o(n^2)}$. Indeed, the fastest known algorithm for edge coloring is obtained by applying the vertex coloring algorithm of Björklund, Husfeldt and Koivisto [2] to the line graph of the input graph. As a result, we get an edge coloring algorithm which, for any graph with m edges and n vertices, runs in time $2^m n^{O(1)}$ and exponential space, which is $2^{\Theta(n^2)}$ for dense graphs. The only progress towards a tailor-made approach for edge coloring is the more recent algorithm of Björklund, Husfeldt, Kaski and Koivisto [1] which still runs in time $2^m n^{O(1)}$ but uses only polynomial space. In this context it is natural to ask for a lower bound. Clearly, any superpolynomial lower bound would imply P≠NP. However, a more feasible goal is to prove a meaningful lower bound under the assumption of a well established conjecture, like Exponential Time Hypothesis (ETH, see Section 2 for a precise formulation). The reduction of Holyer, combined with standard tools (see Section 2) proves that Edge Coloring does not admit an algorithm running in time $2^{o(m)}$ or $2^{o(n)}$, unless ETH fails. At the open problem session of Dagstuhl Seminar 08431 in 2008 [7] it was asked to exclude $2^{O(n)}$ algorithms, assuming ETH. Despite considerable progress in ETH-based lower bounds in recent years [4, 6, 13] this problem stays unsolved (see the report from Dagstuhl Seminar 16451 in 2017 [12]).

List edge coloring is a generalization of edge coloring. An *edge list assignment* $L : E(G) \to 2^{\mathbb{N}}$ is a function that assigns to each edge e of G a set (often called a *list*) $L(e)$ of allowed colors. A function $c : E(G) \to \mathbb{N}$ is a *list edge coloring* of (G, L) if $c(e) \in L(e)$ for every $e \in E(G)$, and $c(e) \neq c(f)$ for every pair of incident edges $e, f \in E(G)$. The notion of list edge coloring is also a frequent topic of research. For example, it is conjectured that if G can be edge colored in k colors for some k, then it can be list edge colored for any edge list assignment with all lists of size at least k. This conjecture has been proved in some classes of graphs like bipartite graphs [8] or planar graphs of maximum degree at least 12 [3].

In this work, we study the computational complexity of list edge coloring. The basic decision problem, List Edge Coloring in Simple Graphs, asks if for a given simple graph G with edge list assignment L there is a list edge coloring of (G, L). Its more general variant, called List Edge Coloring in Multigraphs asks the same question but the input graph does not need to be simple, i.e., it can contain parallel edges. Although the problem seems much more general than Edge Coloring, the two best known algorithms [2, 1] that decide if a given graph admits an edge coloring in k colors solve List Edge Coloring in Multigraphs (and hence also List Edge Coloring in Simple Graphs) within the same time bound, i.e., $2^m m^{O(1)} + O(L)$, where L is the total length of all lists, after only minor modifications (see Proposition 3 in [2]). Multigraphs do not admit any upper bound on the number of edges, hence this time complexity does not translate to a function on n. We show that this is not an accident, because satisfiability of any sufficiently sparse 3-CNF-SAT formula can be efficiently encoded as a list edge coloring instance with a bounded number of vertices. This gives the following result.

▶ **Theorem 1.** *If there is a function $f : \mathbb{N} \to \mathbb{N}$ such that* List Edge Coloring in Multigraphs *can be solved in time $f(n) \cdot m^{O(1)}$ for any input graph on n vertices and m edges, then $P = NP$.*

For simple graphs $m = O(n^2)$ and hence List Edge Coloring in Simple Graphs admits an algorithm running in time $2^{O(n^2)}$. Our main result states that this bound is essentially optimal, assuming ETH.

▶ **Theorem 2.** *If there is an algorithm for* List Edge Coloring in Simple Graphs *that runs in time* $2^{o(n^2)}$*, then Exponential Time Hypothesis fails.*

Our results have twofold consequences for the Edge Coloring problem. First, one may hope that our reductions can inspire a reduction for Edge Coloring. However, it is possible that such a reduction does not exist and researchers may still try to get an algorithm for Edge Coloring running in time $2^{o(n^2)}$. Then we offer a simple way of verifying if a new idea works: if it applies to the list version as well, there is no hope for it.

2 Preliminaries

For an integer k, we denote $[k] = \{0, \ldots, k-1\}$. If I and J are instances of decision problems P and R, respectively, then we say that I and J are *equivalent* if either both I and J are YES-instances of the respective problems, or both are NO-instances. A clause in a CNF-formula is represented by the set of its literals. For two subsets of vertices A, B of a graph $G = (V, E)$ by $E(A, B)$ we denote the set of edges with one endpoint in A and the other in B.

Exponential-Time Hypothesis.

The Exponential Time Hypothesis (ETH) of Impagliazzo et al. [10] states that there exists a constant $c > 0$, such that there is no algorithm solving 3-SAT in time $O(2^{cn})$. During the recent years, ETH became the central conjecture used for proving tight bounds on the complexity of various problems. One of the most important results connected to ETH is the *Sparsification Lemma* [11], which essentially gives a (many-one) reduction from an arbitrary instance of k-SAT to an instance where the number of clauses is linear in the number of variables. The following well-known corollary can be derived by combining ETH with the Sparsification Lemma.

▶ **Theorem 3** (see e.g. Theorem 14.4 in [5])**.** *Unless ETH fails, there is no algorithm for* 3-SAT *that runs in time* $2^{o(n+m)}$*, where* n, m *denote the numbers of variables and clauses, respectively.*

We need the following regularization result of Tovey [14]. Following Tovey, by (3,4)-SAT we call the variant of 3-SAT where each clause of the input formula contains exactly 3 different variables, and each variable occurs in at most 4 clauses.

▶ **Lemma 4** ([14])**.** *Given a* 3-SAT *formula* φ *with* n *variables and* m *clauses one can transform it in polynomial time into an equivalent* (3,4)-SAT *instance* φ' *with* $O(n + m)$ *variables and clauses.*

Theorem 3 and Lemma 4 give the following corollary.

▶ **Corollary 5.** *Unless ETH fails, there is no algorithm for* (3,4)-SAT *that runs in time* $2^{o(n)}$*, where* n *denotes the number of variables of the input formula.*

3 Hardness of List Edge Coloring in Multigraphs

In order to prove Theorems 1 and 2 we show reductions from (3,4)-SAT to List Edge Coloring with strong bounds on the number of vertices in the output instance. The basic idea of both our reductions is to use two colors, denoted by x_i and $\neg x_i$ for every variable x_i so that in every coloring of the output graph the edges colored in x_i or $\neg x_i$ form a single path with alternating colors. Then colors at the edges of this path of fixed parity can encode

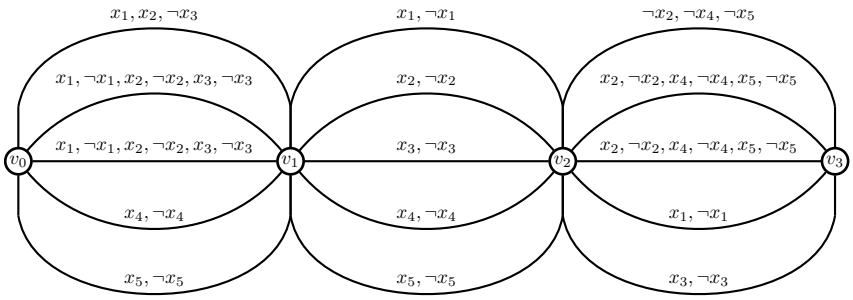


Figure 1 Edges related to clauses $(x_1 \vee x_2 \vee \neg x_3)$ and $(\neg x_2 \vee \neg x_4 \vee \neg x_5)$ assuming that the first of these clauses has color 0 and the second has color 1.

the value of x_i in a satisfying boolean assignment. Moreover, testing a clause $C = \ell_1 \vee \ell_2 \vee \ell_3$ can be done very easily: it suffices to add an edge with the list $\{\ell_1, \ell_2, \ell_3\}$. However this edge can belong to the alternating path of at most one of the three variables in C, and we add two more parallel edges which become elements of the two other alternating paths. Unfortunately, in order to get similar phenomenon in simple graphs, we need to introduce a complicated gadget.

▶ **Lemma 6.** *For any instance φ of* (3,4)-SAT *with n variables there is an equivalent instance (G, L) of* List Edge Coloring in Multigraphs *with 21 vertices and $O(n)$ edges. Moreover, the instance (G, L) can be constructed in polynomial time.*

Proof. Let $\mathrm{vrb}(\varphi)$ and $\mathrm{cls}(\varphi)$ be the sets of variables and clauses of φ, respectively. W.l.o.g. assume $\mathrm{vrb}(\varphi) = \{x_0, \ldots, x_{n-1}\}$.

We construct an auxiliary graph G_φ with $V(G_\varphi) = \mathrm{cls}(\varphi)$ and such that two clauses $C_1, C_2 \in \mathrm{cls}(\varphi)$ are adjacent in G_φ iff $C_1 \cap C_2 \neq \emptyset$. Since every clause has three variables and each variable can belong to at most three other clauses, it follows that the maximum degree of G_φ is at most 9. Let $g : \mathrm{cls}(\varphi) \to [10]$ be the greedy vertex coloring of G_φ in 10 colors, which can be found in linear time in a standard way. For $i \in [10]$, let $\mathcal{C}_i = g^{-1}(i)$.

Let us describe the output instance (G, L). We put $V(G) = \{v_0, \ldots, v_{20}\}$. The edges of G join only vertices of consecutive indices. For every $r \in [10]$, for every clause $C \in \mathcal{C}_r$ we add three new edges with endpoints v_{2r} and v_{2r+1}. The first of these edges, denoted by e^1_C, gets list C, i.e., the three literals of clause C. Let x_i, x_j and x_k be the three variables that appear in C. Then, the two remaining edges, e^2_C and e^3_C, get identical lists of $\{x_i, \neg x_i, x_j, \neg x_j, x_k, \neg x_k\}$. Moreover, for every $r \in [10]$ and for every variable x_i that does not appear in any of the clauses of $\mathcal{C}_r$, we add a new edge $v_{2r}v_{2r+1}$ with list $\{x_i, \neg x_i\}$. Finally, for every $r \in [10]$ and for every variable $x_i \in \mathrm{vrb}(\varphi)$ we add a single new edge $v_{2r+1}v_{2r+2}$ with list $\{x_i, \neg x_i\}$. This finishes the description of the output instance. See Fig. 1 for an example.

In what follows, edges of the form $v_{2r}v_{2r+1}$ are called *positive* and edges of the form $v_{2r+1}v_{2r+2}$ are called *negative*.

▶ **Claim 1.** *For every list edge coloring c of (G, L), for every $i \in [n]$, the edges in $c^{-1}(\{x_i, \neg x_i\})$ form a path $v_0, v_1, \ldots, v_{20}$.*

Proof. For every $r \in [10]$, there is exactly one edge $v_{2r+1}v_{2r+2}$ with list containing x_i or $\neg x_i$, namely with list $\{x_i, \neg x_i\}$. It follows that these 10 edges belong to $c^{-1}(\{x_i, \neg x_i\})$. It suffices to prove that for every $r \in [10]$ there is also exactly one edge $v_{2r}v_{2r+1}$ in $c^{-1}(\{x_i, \neg x_i\})$. This

is clear when x_i does not appear in any of the clauses of $\mathcal{C}_r$, because then there is exactly one edge $v_{2r+1}v_{2r+2}$ with list containing x_i or $\neg x_i$, namely with list $\{x_i, \neg x_i\}$. Otherwise, let $C = \{\ell_i, \ell_j, \ell_k\}$ be the clause of $\mathcal{C}_r$ where $\ell_i \in \{x_i, \neg x_i\}$. Let $\ell_j \in \{x_j, \neg x_j\}$, $\ell_k \in \{x_k, \neg x_k\}$. Then there are exactly three edges e^1_C, e^2_C, e^3_C incident to v_{2r} and v_{2r+1} and with list containing one of literals in the set $\{x_i, \neg x_i, x_j, \neg x_j, x_k, \neg x_k\}$. Indeed, $L(e^1_C) = \{\ell_i, \ell_j, \ell_k\}$, and $L(e^2_C) = L(e^3_C) = \{x_i, \neg x_i, x_j, \neg x_j, x_k, \neg x_k\}$. However, we have already proved that for every $q \in \{i, j, k\}$, one of the edges with endpoints v_{2r+1} and v_{2r+2} is colored with x_q or $\neg x_q$. Hence, since every color class is a matching, for every $q \in \{i, j, k\}$, at most one of the edges in $\{e^1_C, e^2_C, e^3_C\}$ is colored with x_q or $\neg x_q$. However, the lists of e^1_C, e^2_C, e^3_C contain only colors of the form x_q or $\neg x_q$ for $q \in \{i, j, k\}$. It follows that for every $q \in \{i, j, k\}$ exactly one of the edges in $\{e^1_C, e^2_C, e^3_C\}$ is colored with x_q or $\neg x_q$. In particular there is exactly one edge $v_{2r}v_{2r+1}$ in $c^{-1}(\{x_i, \neg x_i\})$. ◀

Since c is an edge coloring, the path from the claim above is colored in one of two ways, either by $x_i, \neg x_i, x_i, \neg x_i, \ldots$, or by $\neg x_i, x_i, \neg x_i, x_i, \ldots$. This implies the following claim.

▶ **Claim 2.** *For every list edge coloring c of (G, L), for every $i \in [n]$, we have $|c^{-1}(x_i)| = |c^{-1}(\neg x_i)| = 10$ and either all edges in $c^{-1}(x_i)$ are positive and all edges in $c^{-1}(\neg x_i)$ are negative or all edges in $c^{-1}(x_i)$ are negative and all edges in $c^{-1}(\neg x_i)$ are positive.*

Now we are ready to prove that φ and (G, L) are equivalent.

Assume c is a list edge coloring of (G, L). Define a boolean assignment $f : \mathrm{vrb}(\varphi) \to \{T, F\}$ by setting x_i to T iff all edges in $c^{-1}(x_i)$ are positive. Now consider an arbitrary clause C. By construction, there is a positive edge e with $L(e) = C$. If $c(e) = x_q$ for some variable x_q then by Claim 2 all edges in $c^{-1}(x_q)$ are positive, and hence $f(x_q) = T$. Since $c(e) \in L(e)$ we have $x_q \in C$, so C is satisfied. If $c(e) = \neg x_q$ for some variable x_q then by Claim 2 all edges in $c^{-1}(x_q)$ are negative and hence $f(x_q) = F$. Again, since $c(e) \in L(e)$ we have $\neg x_q \in C$, so C is satisfied.

Assume φ is satisfiable and let $f : \mathrm{vrb}(\varphi) \to \{T, F\}$ be a satisfying assignment. We define a list edge coloring c of (G, L) as follows. Recall that for every $r \in [10]$, and for every clause $C \in \mathcal{C}_r$ there is an edge e^1_C with $L(e^1_C) = C$ and edges e^2_C, e^3_C with $L(e^2_C) = L(e^3_C) = \{x_i, \neg x_i, x_j, \neg x_j, x_k, \neg x_k\}$, where x_i, x_j and x_k are the three variables that appear in C. We color e^1_C with any of the satisfied literals of C. Without loss of generality, assume $c(e^1_C) \in \{x_i, \neg x_i\}$. Then we color e^2_C with x_j if $f(x_j) = T$ and with $\neg x_j$ otherwise. Similarly, we color e^3_C with x_k if $f(x_k) = T$ and with $\neg x_k$ otherwise. Each of the remaining positive edges e of G has its list equal $\{x_i, \neg x_i\}$ for some $x_i \in \mathrm{vrb}(\varphi)$. We color e with x_i if $f(x_i) = T$ and with $\neg x_i$ otherwise. It follows that every positive edge is colored with a satisfied literal. Every negative edge $\tilde{e}$ has its list equal to $\{x_i, \neg x_i\}$ for some $x_i \in \mathrm{vrb}(\varphi)$. We color $\tilde{e}$ with x_i when $f(x_i) = F$ and with $\neg x_i$ when $f(x_i) = T$. It follows that every negative edge is colored with an unsatisfied literal. Let us show that c does not color incident edges with the same color. Since the lists of parallel negative edges are disjoint, in our coloring there are no parallel negative edges of the same color. Assume there are two parallel positive edges of the form $v_{2r}v_{2r+1}$ of the same color ℓ, for some $r \in [10]$. Then the variable of ℓ belongs to a clause in $\mathcal{C}_r$, for otherwise there is exactly one edge with endpoints $v_{2r}v_{2r+1}$ and with list containing ℓ. However, since $\mathcal{C}_r$ is independent in G_φ, there is exactly one such clause C in $\mathcal{C}_r$. It follows that the two parallel edges are among the three edges e^1_C, e^2_C, e^3_C. However, these three edges got different colors, a contradiction. If two edges are incident but not parallel, one of them is positive and the other negative. The former is colored with a satisfied literal and the latter with an unsatisfied literal, so they are colored differently. Hence c is a proper list edge coloring, as required. This ends the proof of Lemma 6. ◀

Theorem 1 follows immediately from Lemmas 4 and 6 and the NP-hardness of 3-SAT.

4 Hardness of List Edge Coloring in Simple Graphs

This section is devoted to the proof of the following lemma.

▶ **Lemma 7.** *For any instance φ of* (3,4)-SAT *with n variables there is an equivalent instance (G, L) of* List Edge Coloring in Simple Graphs *with $O(\sqrt{n})$ vertices. Moreover, the instance (G, L) can be constructed in polynomial time.*

4.1 Intuition

The general idea is to follow the approach of Lemma 6 and replace the edges with multiplicity $O(n)$ with bipartite graphs with $O(\sqrt{n})$ vertices and $O(n)$ edges. In our construction, for every $r \in [10]$, we replace every two consecutive bundles of parallel edges between v_{2r}, v_{2r+1}, and v_{2r+2} from the construction in Lemma 6 by seven layers L_i, $i = 6r+1, \ldots, 6r+7$, each of $O(\sqrt{n})$ vertices, with edges joining both consecutive and non-consecutive layers. The subgraph induced by $\bigcup_{i=6r+1}^{6r+7} L_i$ is called the r-th *clause verifying gadget* G_r. (Note that the layers L_i for $i \equiv 1 \pmod 6$ are shared between consecutive gadgets.) Analogously as in Lemma 6, the role of G_r is to check whether all clauses in $\mathcal{C}_r$ are satisfied. We add also two additional layers L_0 and L_{62} which make some of our arguments simpler.

4.2 Construction

It will be convenient to assume that $\sqrt{n} \in \mathbb{N}$. We do not lose on generality because otherwise we just add $n^+ = (\lceil\sqrt{n}\rceil + 1)^2 - n$ variables $y_1, y_2, \ldots, y_{n^+}$ and clauses

$$\{y_1, y_2, y_3\}, \{y_2, y_3, y_4\}, \ldots, \{y_{n^+-2}, y_{n^+-1}, y_{n^+}\}.$$

Note that $n^+ \geq 3$, $n^+ \leq (\sqrt{n}+2)^2 - n = 4\sqrt{n} + 4$ and $\sqrt{n+n^+} = \lceil\sqrt{n}\rceil + 1 \in \mathbb{N}$. Hence we added only $O(\sqrt{n})$ variables and clauses, and the resulting formula is still a (3,4)-SAT instance.

We begin as in Lemma 6, by building the graph G_φ, and finding its greedy coloring g which partitions the clause set into 10 color classes $\mathcal{C}_r$, $r \in [10]$. Let us build the instance (G, L) step by step.

Add two sets of vertices (called *layers*) $L_i = \{v_j^i \mid j \in [\sqrt{n}]\}$, $i = 0, 1$. Then add all possible n edges between L_0 and L_1 forming a complete bipartite graph. Map the n variables to the n edges in a $1-1$ way. For every $i \in [n]$, set the list of the edge assigned to x_i to $\{x_i, \neg x_i\}$.

The vertex set $V(G)$ contains 60 more layers of vertices L_i, $i = \{2, \ldots, 61\}$, where $L_i = \{v_j^i \mid j \in [6\sqrt{n}+3]\}$. Finally, $L_{62} = \{v_j^{62} \mid j \in [\sqrt{n}+1]\}$. Also denote $L_{-1} = L_{63} = \emptyset$. In what follows we add the remaining edges of G. Whenever we add edges between L_i and L_{i-1}, for every $j < i$ all the edges of the output graph between L_j and L_{j-1} are already added. We will make sure to keep the following invariants satisfied during the process of construction (note that they hold for the part constructed so far).

▶ **Invariant 1 (Uniqueness).** *For every $i \in [62]$, for every variable $x_j \in \mathrm{vrb}(\varphi)$ there is at most one edge $uv \in E(L_i, L_{i+1})$ such that $\{x_j, \neg x_j\} \cap L(uv) \neq \emptyset$. Moreover, after finishing adding edges between L_i and L_{i+1}, there is exactly one such edge.*

Using the notation from Invariant 1, if the edge uv exists, we can denote $v_{i,j}^+ = u$ and $v_{i+1,j}^- = v$.

▶ **Invariant 2 (Flow).** *For every $i \in \{1, \ldots, 62\}$, for every variable $x_j \in \mathrm{vrb}(\varphi)$ we have that $v^-_{i,j} = v^+_{i,j}$, unless $v^-_{i,j}$ or $v^+_{i,j}$ is undefined. Moreover, the equality holds after finishing adding edges between L_i and L_{i+1}.*

Thanks to Invariant 2, after finishing adding edges between L_i and L_{i+1}, we can just define $v_{i,j} := v^-_{i,j} = v^+_{i,j}$ for $i \in \{1, \ldots, 61\}$. We also put $v_{0,j} = v^+_{0,j}$ and $v_{62,j} = v^-_{62,j}$. In our construction we will use some additional colors apart from the literals. The following invariant holds.

▶ **Invariant 3 (Lists).** *For every edge e of G, the list $L(e)$ contains at least one literal.*

For every $i \in [62]$, for every vertex $v \in L_i$, let $\deg^-(v) = |E(L_{i-1}, \{v\})|$ and $\deg^+(v) = |E(L_{i+1}, \{v\})|$.

▶ **Invariant 4 (Indegrees).** *For every $i \in [62]$ for vertex $v \in L_i$ we have $\deg^-(v) \leq \sqrt{n}$.*

▶ **Invariant 5 (Jumping edges).** *For every $i \in [62]$, for vertex $v \in L_i$ there are at most $\sqrt{n}$ edges from v to layers L_j for $j > i + 1$.*

By Invariant 2 and Invariant 3, for every vertex $v \in V(G)$ it holds that $\deg^+(v) \leq \deg^-(v)$. Hence Invariant 4 gives the claim below.

▶ **Claim 3 (Outdegrees).** *For every $i \in [62]$, for every vertex $v \in L_i$, we have $\deg^+(v) \leq \sqrt{n}$.*

Invariants 1 and 3 immediately imply the following.

▶ **Claim 4.** *For every $i \in [62]$, we have $|E(L_i, L_{i+1})| \leq n$.*

Let us fix $r \in [10]$. We add the edges of the r-th *clause verifying gadget* G_r. Although G is undirected, we will say that an edge uv between L_i and L_j for $i < j$ is *from u to v* and *from L_i to L_j*. Below we *describe* the edges in G_r in the order which is convenient for the exposition. However, the *algorithm* adds the edges between layers in the left-to-right order, i.e., for $i < j$, edges to L_i are added before edges to L_j.

1. Edges to L_ℓ for $\ell = 6r + 2, 6r + 4, 6r + 6$.
   For every clause $C \in \mathcal{C}_r$ we do the following. Let $x_{i_1}, x_{i_2}, x_{i_3}$ be the three different variables that appear in the literals of C. Let $v_j = v^-_{\ell-1,i_j}$ for $j = 1, 2, 3$. Note that vertices v_1, v_2, v_3 need not be distinct. By Claim 3, $|N(v_j) \cap L_\ell| \leq \sqrt{n}$ for $j = 1, 2, 3$. Let $S = \{v \in L_\ell \mid \deg^-(v) = \sqrt{n}\}$. By Claim 4, $|S| \leq \sqrt{n}$. Hence, for $j = 1, 2, 3$ we have $|L_\ell \setminus (N(\{v_j\}) \cup S)| \geq 4\sqrt{n} + 3$ and we can pick a vertex $w_j \in L_\ell$ that has at most $\sqrt{n} - 1$ edges from $L_{\ell-1}$, is not adjacent to v_j, and is different than $w_{j'}$ for each $j' < j$. If $\ell = 6k + 6$ we additionally require that for every $j = 1, 2, 3$, the vertex w_j is not adjacent to v^-_{6r+2,i_j} or v^-_{6r+4,i_j}. By Invariant 5 this eliminates at most $2\sqrt{n}$ more candidates, so it is still possible to choose all the w_j's. For each $j = 1, 2, 3$, we add an edge $v_j w_j$ with $L(v_j w_j) = \{x_{i_j}, \neg x_{i_j}\}$. Moreover, if $\ell = 6k + 6$, for every $j = 1, 2, 3$ we add an edge $v^-_{6r+2,i_j} w_j$ with list $\{x_{i_j}, \neg x_{i_j}, a_{i,j}\}$ and an edge $v^-_{6r+4,i_j} w_j$ with list $\{x_{i_j}, \neg x_{i_j}, b_{i,j}\}$. The conditions used to choose w_1, w_2 and w_3 guarantee that we do not introduce parallel edges.
   For every variable x_i that is not present in any of the clauses of $\mathcal{C}_r$ we find a vertex $w \in L_\ell$ that has at most $\sqrt{n} - 1$ edges from $L_{\ell-1}$ and is not adjacent to $v^-_{\ell-1,i}$. Again, this is possible because there are at most $2\sqrt{n}$ vertices in L_ℓ that violate any of these constraints. We add an edge $v^-_{\ell-1,i} w$ with $L(v^-_{\ell-1,i} w) = \{x_i, \neg x_i\}$.
   Note that all invariants are satisfied: for Invariant 1 it follows from the fact that $\mathcal{C}_r$ is independent in G_φ, while invariants 2, 3, 4 follow immediately from the construction.

Invariant 5 stays satisfied after adding $v^-_{6r+2,i_j}w_j$ because for every variable x_k such that $v^-_{6r+2,k} = v^-_{6r+2,i_j}$ we add at most one edge from v^-_{6r+2,i_j} to L_{6r+6}, and the number of such variables is equal to $\deg^-(v^-_{6r+2,i_j})$, which is at most $\sqrt{n}$ by Invariant 4 (analogous argument applies to adding the edge $v^-_{6r+4,i_j}w_j$).

2. Edges to L_ℓ for $\ell = 6r+3, 6r+5, 6r+7$.
   For every clause $C \in \mathcal{C}_r$ we do the following. Let $C = \{\ell_1, \ell_2, \ell_3\}$ and let x_{i_j} be the variable from the literal of ℓ_j, for $j = 1, 2, 3$. Let $w_j = v^-_{\ell-1,i_j}$ for $j = 1, 2, 3$. By Claim 3, $|N(\{w_1, w_2, w_3\} \cap L_\ell)| \leq 3\sqrt{n}$. Also, there are at most $\sqrt{n}+2$ vertices in L_ℓ with at least $\sqrt{n}-2$ edges from $L_{\ell-1}$. Indeed, otherwise $|E(L_{\ell-1}, L_\ell)| \geq n + \sqrt{n} - 6$ and either $n \leq 36$ (and the lemma is trivial) or there is a contradiction with Claim 4. Hence, we can find a vertex $z_{\ell,C} \in L_\ell$ that has at most $\sqrt{n}-3$ edges to $L_{\ell-1}$ and is not adjacent to $\{w_1, w_2, w_3\}$. If $\ell = 6k+7$ we additionally require that the vertex $z_{6k+7,C}$ is not adjacent to $z_{6k+3,C}$ or $z_{6k+5,C}$. By Invariant 5 this eliminates at most $2\sqrt{n}$ more candidates, so it is still possible to choose vertex $z_{6k+7,C}$. For each $j = 1, 2, 3$, we add an edge $w_j z_{\ell,C}$. We put $L(w_j z_{6r+3,C}) = \{x_{i_j}, \neg x_{i_j}, a_{i_j}\}$, $L(w_j z_{6r+5,C}) = \{x_{i_j}, \neg x_{i_j}, b_{i_j}\}$, and $L(w_j z_{6r+7,C}) = \{\ell_j, c_C, d_C\}$. (The colors $a_{i_j}, b_{i_j}, c_C, d_C$ are not literals — these are new auxiliary colors; each variable x_i has its own distinct auxiliary colors a_i, b_i, and each clause C has its own auxiliary colors c_C, d_C.) We add edges $z_{6r+3,C}z_{6r+7,C}$ and $z_{6r+5,C}z_{6r+7,C}$, both with lists $\{x_{i_1}, \neg x_{i_1}, x_{i_2}, \neg x_{i_2}, x_{i_3}, \neg x_{i_3}\}$.
   For every variable x_i that is not present in any of the clauses of $\mathcal{C}_r$ we proceed analogously as in Step 1.
   The invariants hold for the similar reasons as before. In particular, Invariant 5 stays satisfied after adding $z_{6r+3,C}z_{6r+7,C}$ because for every clause C' such that $z_{6r+3,C'} = z_{6r+3,C}$ we add exactly one edge from $z_{6r+3,C}$ to L_{6r+7}, and the number of such clauses is bounded by $\deg^-(z_{6r+3,C})/3$, which is at most $\sqrt{n}/3$ by Invariant 4 (analogous argument applies to adding the edge $z_{6r+5,C}z_{6r+7,C}$).

Finally, we add edges between L_{61} and L_{62}. For every variable x_i we find a vertex $w \in L_{62}$ that is not adjacent to $v^-_{61,i}$, which is possible because $\deg^+(v^-_{61,i}) \leq \sqrt{n}$. We add an edge $v^-_{61,i}w$ with $L(v^-_{61,i}w) = \{x_i, \neg x_i\}$.

The following claims follow directly from the construction.

▶ **Claim 5.** *For every $r \in [10]$, for every clause $C \in \mathcal{C}_r$ with variables $x_{i_1}, x_{i_2}, x_{i_3}$, and for each $\ell = 6r+3, 6r+5, 6r+7$ we have $v_{\ell,i_1} = v_{\ell,i_2} = v_{\ell,i_3} = z_{\ell,C}$. Moreover, for each $\ell = 6r+3, 6r+5, 6r+7$ and $j = 1, 2, 3$ we have $L(z_{\ell,C}v_{\ell+1,i_j}) = \{x_{i_j}, \neg x_{i_j}\}$.*

▶ **Claim 6.** *For every edge $uv \in E(G)$, where $u \in L_j, v \in L_k$, if $\{x_i, \neg x_i\} \cap L(uv) \neq \emptyset$, then $u = v_{j,i}$ and $u = v_{k,i}$.*

This finishes the description of the output instance. Since G contains $O(1)$ layers, each with $O(\sqrt{n})$ vertices, it follows that $|V(G)| = O(\sqrt{n})$, as required. See Fig 2 for an illustration of edges representing a single clause within a clause verifying gadget.

4.3 Structure of coloring

Similarly to multigraphs the crux of the equivalence between instances is the following claim.

▶ **Claim 7.** *For every list edge coloring c of (G, L), for every $i \in [n]$, the edges in $c^{-1}(\{x_i, \neg x_i\})$ form a path P_i from L_0 to L_{62}. Moreover, if P_i contains an edge $v_{6r+6,i}v_{6r+7,i}$ for some $r \in [10]$, then this edge is preceded by an even number of edges on P_i.*

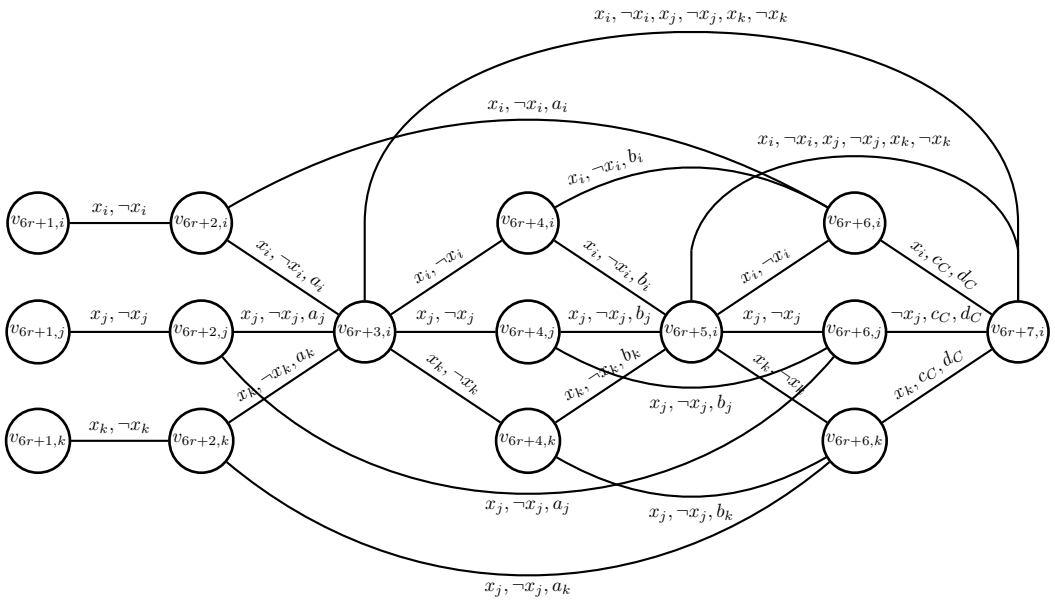


Figure 2 Edges in the gadget G_r related to a clause $(x_i \vee \neg x_j \vee x_k)$ from $\mathcal{C}_r$.

Proof. Fix $i \in [n]$. For convenience, denote $E_i = c^{-1}(\{x_i, \neg x_i\})$. By Invariant 1 there is exactly one edge between L_0 and L_1 that has x_i or $\neg x_i$ on its list, namely $v_{0,i}v_{1,i}$. Similarly, there is exactly one edge between L_{61} and L_{62} that has x_i or $\neg x_i$ on its list, namely $v_{61,i}v_{62,i}$. Since $L(v_{0,i}v_{1,i}) = L(v_{61,i}v_{62,i}) = \{x_i, \neg x_i\}$, we know that $v_{0,i}v_{1,i}, v_{61,i}v_{62,i} \in E_i$, and these are the only edges of E_i in $E(L_0, L_1) \cup E(L_{61}, L_{62})$. Observe that edges between non-consecutive layers never leave the clause verifying gadgets. Hence, for the first part of the claim, it suffices to show that for every $r \in [10]$, the edges in $E_i \cap E(G_r)$ form a path between $v_{6r+1,i}$ and $v_{6r+7,i}$. In fact, by Claim 6 it suffices to show that $E_i \cap E(G_r)$ *contains* a path between $v_{6r+1,i}$ and $v_{6r+7,i}$ that visits all the vertices $\{v_{6r+j,i} \mid j = 1, \ldots, 7\}$. To this end, fix $r \in [10]$.

First assume that x_i does not appear in any clause of $\mathcal{C}_r$. Then G_r contains the path $v_{6r+1,i}, v_{6r+2,i}, \ldots, v_{6r+7,i}$, where each edge has the list $\{x_i, \neg x_i\}$. It immediately implies that all edges of this path are in $E_i \cap E(G_r)$, as required.

Now let us assume that x_i appears in a clause $C \in \mathcal{C}_r$. Let $C = \{\ell_i, \ell_j, \ell_k\}$ and assume that the literal ℓ_i contains x_i, the literal ℓ_j contains a variable x_j, and the literal ℓ_k contains a variable x_k. Observe that for $j = 1, 3, 5$ we have $v_{6r+j,i}v_{6r+j+1,i} \in E_i$ because these edges have their lists equal to $\{x_i, \neg x_i\}$. Note also that $\Delta(E_i) \leq 2$ because E_i is a union of two matchings (colors). We consider three subcases.

1. Assume $v_{6r+3,i}v_{6r+7,i} \in E_i$. Since $\Delta(E_i) \leq 2$ and $v_{6r+3,i}v_{6r+4,i} \in E_i$ we know that $v_{6r+2,i}v_{6r+3,i} \notin E_i$, and as a consequence, $c(v_{6r+2,i}v_{6r+3,i}) = a_i$. Hence $c(v_{6r+2,i}v_{6r+6,i}) \neq a_i$, which implies that $v_{6r+2,i}v_{6r+6,i} \in E_i$. Then, since $\Delta(E_i) \leq 2$ and $v_{6r+5,i}v_{6r+6,i} \in E_i$ we know that $v_{6r+4,i}v_{6r+6,i} \notin E_i$, and as a consequence, $c(v_{6r+4,i}v_{6r+6,i}) = b_i$. Hence $c(v_{6r+4,i}v_{6r+5,i}) \neq b_i$, which implies that $v_{6r+4,i}v_{6r+5,i} \in E_i$. Thus, we have shown that E_i contains the path $v_{6r+1,i}, v_{6r+2,i}, v_{6r+6,i}, v_{6r+5,i}, v_{6r+4,i}, v_{6r+3,i}, v_{6r+7,i}$, as required.
2. Assume $v_{6r+5,i}v_{6r+7,i} \in E_i$. Since $\Delta(E_i) \leq 2$ and $v_{6r+5,i}v_{6r+6,i} \in E_i$ we know that $v_{6r+4,i}v_{6r+5,i} \notin E_i$, and as a consequence, $c(v_{6r+4,i}v_{6r+5,i}) = b_i$. Hence $c(v_{6r+4,i}v_{6r+6,i}) \neq b_i$, which implies that $v_{6r+4,i}v_{6r+6,i} \in E_i$. Then, since $\Delta(E_i) \leq 2$ and $v_{6r+5,i}v_{6r+6,i} \in E_i$

we know that $v_{6r+2,i}v_{6r+6,i} \notin E_i$, and as a consequence, $c(v_{6r+2,i}v_{6r+6,i}) = a_i$. Hence $c(v_{6r+2,i}v_{6r+3,i}) \neq a_i$, which implies that $v_{6r+2,i}v_{6r+3,i} \in E_i$. Thus, we have shown that E_i contains the path $v_{6r+1,i}, v_{6r+2,i}, v_{6r+3,i}, v_{6r+4,i}, v_{6r+6,i}, v_{6r+5,i}, v_{6r+7,i}$, as required.

3. Assume $v_{6r+3,i}v_{6r+7,i}, v_{6r+5,i}v_{6r+7,i} \notin E_i$. Since $L(v_{6r+3,i}v_{6r+7,i}) = L(v_{6r+5,i}v_{6r+7,i}) = \{x_i, \neg x_i, x_j, \neg x_j, x_k, \neg x_k\}$ we infer that $v_{6r+3,i}v_{6r+7,i}, v_{6r+5,i}v_{6r+7,i} \in E_j \cup E_k$. By Claim 5 we know that $v_{6r+7,i} = v_{6r+7,j} = v_{6r+7,k}$, $v_{6r+7,i}v_{6r+8,j} \in E_j$ and $v_{6r+7,i}v_{6r+8,k} \in E_k$. Since $\Delta(E_j) \leq 2$ and $\Delta(E_k) \leq 2$, we get that $v_{6r+3,i}v_{6r+7,i} \in E_j$ and $v_{6r+5,i}v_{6r+7,i} \in E_k$ or vice versa. In any case, $v_{6k+6,j}, v_{6k+7,i} \notin E_j$, and $v_{6k+6,k}, v_{6k+7,i} \notin E_k$. Recall that $L(v_{6k+6,j}, v_{6k+7,i}) = \{\ell_j, c_C, d_C\}$ and $L(v_{6k+6,k}, v_{6k+7,i}) = \{\ell_k, c_C, d_C\}$. It follows that $c(\{v_{6k+6,j}v_{6k+7,i}, v_{6k+6,k}v_{6k+7,i}\}) = \{c_C, d_C\}$. Then $c(v_{6k+6,i}, v_{6k+7,i}) \notin \{c_C, d_C\}$. Since $L(v_{6k+6,i}, v_{6k+7,i}) = \{\ell_i, c_C, d_C\}$, we get that $v_{6k+6,i}, v_{6k+7,i} \in E_i$. Then, since $\Delta(E_i) \leq 2$ and $v_{6r+5,i}v_{6r+6,i} \in E_i$ we know that $v_{6r+2,i}v_{6r+6,i}, v_{6r+4,i}v_{6r+6,i} \notin E_i$, and as a consequence, $c(v_{6r+2,i}v_{6r+6,i}) = a_i$ and $c(v_{6r+4,i}v_{6r+6,i}) = b_i$. Hence $c(v_{6r+2,i}v_{6r+3,i}) \neq a_i$, and $c(v_{6r+4,i}v_{6r+5,i}) \neq b_i$ which implies that $v_{6r+2,i}v_{6r+3,i}, v_{6r+4,i}v_{6r+5,i} \in E_i$. Thus, E_i contains the path $v_{6r+1,i}, v_{6r+2,i}, v_{6r+3,i}, v_{6r+4,i}, v_{6r+5,i}, v_{6r+6,i}, v_{6r+7,i}$, as required.

For the second part of the claim recall that P_i decomposes into an edge from L_0 to L_1, 10 paths of length 6 inside the gadgets and an edge from L_{61} to L_{62}. Moreover, if P_i contains an edge $v_{6r+6,i}v_{6r+7,i}$ for some $r \in [10]$, then this edge is the last edge of one of the 10 paths of length 6. It follows that it is preceded by $1 + 6r + 5$ edges, which is an even number. ◀

4.4 Equivalence

Assume c is a list edge coloring of (G, L). Define a boolean assignment $f : \mathrm{vrb}(\varphi) \to \{T, F\}$ by setting x_i to T iff the first edge of the path P_i from Claim 7 is colored by x_i. Note that P_i is colored alternately with x_i and $\neg x_i$ and every odd edge on P_i (i.e., preceded by an even number of edges) is colored with a satisfied literal. Now consider an arbitrary clause C. Let $r = g(C)$. Let $C = \{\ell_1, \ell_2, \ell_3\}$ and let x_{i_j} be the variable from the literal of ℓ_j, for $j = 1, 2, 3$. By construction, there are three edges $v_{6r+6,i_j}z_{6r+7,C}$, for $j = 1, 2, 3$ with $L(v_{6r+6,i_j}z_{6r+7,C}) = \{\ell_j, c_C, d_C\}$. At most two of these edges are colored with c_C or d_C, so there is $j = 1, 2, 3$ such that $c(v_{6r+6,i_j}z_{6r+7,C}) = \ell_j$. In particular, $v_{6r+6,i_j}z_{6r+7,C} \in c^{-1}(\{x_{i_j}, \neg x_{i_j}\})$ and hence, by Claim 7 we know that $v_{6r+6,i_j}z_{6r+7,C} \in P_{i_j}$. However, by the second part of Claim 7 this edge is preceded by an even number of edges on P_{i_j}. It follows that ℓ_j is satisfied.

Assume φ is satisfiable and let $f : \mathrm{vrb}(\varphi) \to \{T, F\}$ be a satisfying assignment. We define a list edge coloring c of (G, L) as follows. Consider any edge $e \in E(L_0, L_1)$. Then $L(e) = \{x_i, \neg x_i\}$. We color e with x_i when $f(x_i) = T$ and with $\neg x_i$ otherwise. Now consider any edge $e \in E(L_{61}, L_{62})$. Again $L(e) = \{x_i, \neg x_i\}$. We color e with x_i when $f(x_i) = F$ and with $\neg x_i$ otherwise. By Invariant 1 incident edges get different colors in the partial coloring described so far. In what follows we describe $c|_{E(G_r)}$ for every $r \in [10]$ separately. Fix $r \in [10]$.

Consider an arbitrary clause $C \in \mathcal{C}_r$. Let $C = \{\ell_1, \ell_2, \ell_3\}$ and let x_{i_j} be the variable from the literal of ℓ_j, for $j = 1, 2, 3$. Since φ is satisfied by f, at least one literal of C is satisfied by f, by symmetry we can assume it is ℓ_1. Consider the three edge disjoint paths

$$R_1 = v_{6r+1,i_1}, v_{6r+2,i_1}, v_{6r+3,i_1}, v_{6r+4,i_1}, v_{6r+5,i_1}, v_{6r+6,i_1}, v_{6r+7,i_1},$$

$$R_2 = v_{6r+1,i_2}, v_{6r+2,i_2}, v_{6r+6,i_2}, v_{6r+5,i_2}, v_{6r+4,i_2}, v_{6r+3,i_2}, v_{6r+7,i_2},$$

$$R_3 = v_{6r+1,i_3}, v_{6r+2,i_3}, v_{6r+3,i_3}, v_{6r+4,i_3}, v_{6r+6,i_3}, v_{6r+5,i_3}, v_{6r+7,i_3}.$$

For each $j = 1, 2, 3$ the path R_j is colored by x_{i_j} and $\neg x_{i_j}$ alternately, beginning with $\neg x_{i_j}$ if $f(x_{i_j}) = T$ and with x_{i_j} if $f(x_{i_j}) = F$. Note that edges of R_1, R_2 and R_3 are colored by

colors from their lists. Indeed, this is obvious for every edge apart from $v_{6r+6,i_1}, v_{6r+7,i_1}$, because their lists contain $\{x_{i_j}, \neg x_{i_j}\}$. Edge $v_{6r+6,i_1}, v_{6r+7,i_1}$ is colored with x_{i_j} if $f(x_{i_j}) = T$ and with $\neg x_{i_j}$ if $f(x_{i_j}) = F$. It follows that $v_{6r+6,i_1}, v_{6r+7,i_1}$ is colored with the literal from $\{x_{i_1}, \neg x_{i_1}\}$ which is satisfied by f, hence it is colored by ℓ_1, and $\ell_1 \in L(v_{6r+6,i_1}, v_{6r+7,i_1})$, as required. Finally, we put

$$
\begin{aligned}
c(v_{6r+2,i_1}v_{6r+6,i_1}) &= a_{i_1},\\
c(v_{6r+4,i_1}v_{6r+6,i_1}) &= b_{i_1},\\
c(v_{6r+2,i_2}v_{6r+3,i_2}) &= a_{i_2},\\
c(v_{6r+4,i_2}v_{6r+6,i_2}) &= b_{i_2},\\
c(v_{6r+2,i_3}v_{6r+6,i_3}) &= a_{i_3},\\
c(v_{6r+4,i_3}v_{6r+5,i_3}) &= b_{i_3},\\
c(v_{6r+6,i_2}v_{6r+7,i_2}) &= c_C,\\
c(v_{6r+6,i_3}v_{6r+7,i_3}) &= d_C.
\end{aligned}
$$

Thus we have colored all edges of G_r which have lists containing a variable from C.

Now consider any variable x_i that does not appear in any clause of $\mathcal{C}_r$. Consider the path $v_{6r+1,i}, v_{6r+2,i}, \ldots, v_{6r+7,i}$. If $f(x_i) = T$, color the path with the sequence of colors $\neg x_i, x_i, \neg x_i, \ldots, x_i$, and otherwise with the sequence of colors $x_i, \neg x_i, x_i, \ldots, \neg x_i$.

Thus we have colored all the edges of G_r. It is straightforward to check that for every $r \in [10]$ the subgraph G_r is colored properly. It remains to show that vertices in the layers L_i for $i \equiv 1 \pmod 6$ are not incident to two edges of the same color. Clearly, this cannot happen for colors a_j or b_j for any $j \in [n]$, because they are not present on lists of edges incident to L_i for $i \equiv 1 \pmod 6$. Also, it cannot happen for colors c_C or d_C for any clause C, because edges with these colors on their list only join L_{i-1} with L_i for $i \equiv 1 \pmod 6$, so two incident edges colored with c_C or d_C cannot belong to different gadgets. Finally, consider colors $\{x_i, \neg x_i\}$ for a fixed $i \in [n]$. The edges with these colors form a path of length 62, starting with $v_{0,i}v_{1,i}$, and continued as follows. The edge $v_{0,i}v_{1,i}$ is followed by 10 paths of length 6. For every $r \in [10]$, the r-th path of length 10 begins in $v_{6r+1,i}$ and ends in $v_{6r+7,i} = v_{6(r+1)+1,i}$. Finally, the 62-path ends with edge $v_{61,i}v_{62,i}$. Note that $v_{0,i}v_{1,i}$ is colored with the satisfied literal. Next, for every $r \in [10]$, the first edge of the r-th 10-path is colored with the non-satisfied literal and its last edge is colored by the satisfied literal. Finally, $v_{61,i}v_{62,i}$ is colored with the non-satisfied literal. It follows that the 62-path of all edges with colors from $\{x_i, \neg x_i\}$ is colored alternately in x_i and $\neg x_i$, as required. This finishes the proof that c is a list edge coloring of (G, L), and the proof of Lemma 7.

4.5 Proof of Theorem 2

Theorem 2 follows immediately from Lemma 7 and Corollary 5. Indeed, if there is an algorithm A which solves List Edge Coloring in Simple Graphs in time $2^{o(|V(G)|^2)}$, then by Lemma 7 an n-variable instance of (3,4)-SAT can be transformed to a $O(\sqrt{n})$-vertex instance of List Edge Coloring in Simple Graphs in polynomial time and next solved in time $2^{o(n)}$ using A, which contradicts ETH by Corollary 5.

5 Conclusions and further research

In this work we have shown that List Edge Coloring in Simple Graphs does not admit an algorithm that runs in time $2^{o(n^2)}$, unless ETH fails. This has consequences for designing algorithms for Edge Coloring: in order to break the barrier $2^{O(n^2)}$ one has to use methods

that exploit symmetries between colors, and in particular do not apply to the list version. On the other hand, one may hope that our reductions can inspire a reduction to Edge Coloring which would exclude at least a $2^{O(n)}$-time algorithm. However it seems that Edge Coloring requires a significantly different approach. In our reductions we were able to encode information (namely, the boolean value of a variable in a satisfying assignment) in a *color* of an edge. In the case of Edge Coloring this is not possible, because one can recolor any edge e by choosing an arbitrary different color c' and swapping c' and the color c of e on the maximal path/cycle that contains e and has edges colored with c and c' only.

References

1. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.
2. Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.
3. O. V. Borodin, A. V. Kostochka, and D. R. Woodall. List edge and list total colourings of multigraphs. *J. of Comb. Theory, Ser. B*, 71:184–204, 1997.
4. Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.
5. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
6. Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016. doi:10.1137/130947076.
7. Fedor V. Fomin, Kazuo Iwama, Dieter Kratsch, Petteri Kaski, Mikko Koivisto, Lukasz Kowalik, Yoshio Okamoto, Johan van Rooij, and Ryan Williams. 08431 open problems – moderately exponential time algorithms. In Fedor V. Fomin, Kazuo Iwama, and Dieter Kratsch, editors, *Moderately Exponential Time Algorithms*, number 08431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. URL: http://drops.dagstuhl.de/opus/volltexte/2008/1798.
8. Fred Galvin. The list chromatic index of a bipartite multigraph. *J. Comb. Theory, Ser. B*, 63(1):153–158, 1995. doi:10.1006/jctb.1995.1011.
9. Ian Holyer. The np-completeness of some edge-partition problems. *SIAM J. Comput.*, 10(4):713–717, 1981. doi:10.1137/0210054.
10. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
11. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
12. Moshe Lewenstein, Seth Pettie, and Virginia Vassilevska Williams. Structure and hardness in P (dagstuhl seminar 16451). *Dagstuhl Reports*, 6(11):1–34, 2016. doi:10.4230/DagRep.6.11.1.
13. Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
14. Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Appl. Math.*, 8(1):85–89, 1984.

Load Thresholds for Cuckoo Hashing with Double Hashing

Michael Mitzenmacher[1]
Harvard University, School of Engineering and Applied Sciences, Cambridge, USA
michaelm@eecs.harvard.edu
https://orcid.org/0000-0001-5430-5457

Konstantinos Panagiotou[2]
University of Munich, Institute for Mathematics, Germany
kpanagio@math.lmu.de

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de
https://orcid.org/0000-0002-6477-0106

Abstract

In *k-ary cuckoo hashing*, each of cn objects is associated with k random buckets in a hash table of size n. An *ℓ-orientation* is an assignment of objects to associated buckets such that each bucket receives at most ℓ objects. Several works have determined *load thresholds* $c^* = c^*(k, \ell)$ for k-ary cuckoo hashing; that is, for $c < c^*$ an ℓ-orientation exists with high probability, and for $c > c^*$ no ℓ-orientation exists with high probability.

A natural variant of k-ary cuckoo hashing utilizes *double hashing*, where, when the buckets are numbered $0, 1, \ldots, n-1$, the k choices of random buckets form an arithmetic progression modulo n. Double hashing simplifies implementation and requires less randomness, and it has been shown that double hashing has the same behavior as fully random hashing in several other data structures that similarly use multiple hashes for each object. Interestingly, previous work has come close to but has not fully shown that the load threshold for k-ary cuckoo hashing is the same when using double hashing as when using fully random hashing. Specifically, previous work has shown that the thresholds for both settings coincide, except that for double hashing it was possible that $o(n)$ objects would have been left unplaced. Here we close this open question by showing the thresholds are indeed the same, by providing a combinatorial argument that reconciles this stubborn difference.

2012 ACM Subject Classification Theory of computation → Bloom filters and hashing

Keywords and phrases Cuckoo Hashing, Double Hashing, Load Thresholds, Hypergraph Orientability

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.29

Acknowledgements Part of this work was developed in the Dagstuhl Seminar on Theory and Applications of Hashing in May 2017.

[1] The author was supported in part by NSF grants CCF-1563710, CCF-1535795, CCF-1320231, and CNS-1228598. Part of this work was done while visiting Microsoft Research.
[2] This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programm (grant agreement n° 772606).

1 Introduction

1.1 The Threshold Question

Cuckoo hashing, introduced by Pagh and Rodler [13] and generalized in many subsequent works (see e.g. [1, 2], and [9] for additional background and references), has proven useful both as a theoretical building block and in practical systems. In *k-ary cuckoo hashing*, each of cn objects is associated with k random buckets in a hash table of size n. An *ℓ-orientation* is an assignment of objects to associated buckets such that each bucket receives at most ℓ objects. Several works have determined *load thresholds* $c^* = c^*(k, \ell)$ for k-ary cuckoo hashing; that is, for $c < c^*$ an ℓ-orientation exists with high probability, and for $c > c^*$ no ℓ-orientation exists with high probability. Beyond their theoretical interest, these load thresholds are important for designing systems that use cuckoo hashing, as they provide an accurate guide to what loads can be achieved in practical settings.

A natural variant of k-ary cuckoo hashing utilizes *double hashing*. Double hashing originally appeared in the context of open-address hash tables, where an object j would be placed by successively trying to find an open bucket at locations $h(i, j) = (h_1(j) + ih_2(j)) \bmod |T|$ for $i = 0, 1, \ldots$, where here $|T|$ represents the table size and h_1 and h_2 are two independently selected hash functions. In the context of cuckoo hashing when the buckets are numbered $0, 1, \ldots, n-1$, the k choices of random buckets are of the form $h(i, j) = (h_1(j) + ih_2(j)) \bmod n$ for $i = 0, \ldots, k-1$, so that the choices form an arithmetic progression modulo n.

Double hashing both simplifies implementation and requires less randomness. Moreover, a classical result in the theory of open-address hash tables is that double hashing yields asymptotically the same cost for an unsuccessful search as using full randomness [8], showing that there is negligible performance cost in using double hashing. This type of result, that using double hashing does not change the performance, has since been shown for other hashing-based data structures using several choices, such as Bloom filters [4] and balanced allocation hash tables [10, 11]. We therefore expect that the load thresholds for cuckoo hashing would be the same using double hashing as when using full randomness. Indeed, as we describe in more detail below, previous work has *almost* shown that the thresholds are the same, but completing the argument has proven stubbornly elusive. Here we complete the proof through a suitable combinatorial argument.

1.2 Terminology

In the rest of the paper, we make use of the following terminology.

Fully random graph. Let $H^k_{n,cn}$ be a k-uniform random hypergraph with vertex set $\mathbb{Z}_n$ and cn edges or something "morally equivalent". Specifically, for our purposes it is often convenient to have perfect independence of edges, each edge e being picked as $e = \{x_1, \ldots, x_k\}$ where $x_1, \ldots, x_k$ are chosen independently and uniformly from $\mathbb{Z}_n$. Note that this may result in some edges of size less than k, as well as duplicate edges. These deviations do not change the threshold, as is known via standard arguments.

Double-Hashing graph. Similarly, $D^k_{n,cn}$ is also a random hypergraph with vertex set $\mathbb{Z}_n$, but the cn edges must be k-term arithmetic progressions. More precisely, each edge e is independently sampled as $e = \{a + ib \bmod n \mid 0 \le i < k\}$ for $a \in \mathbb{Z}_n$ and $1 \le b < n/2$ chosen independently and uniformly at random. If n is prime then each edge has size k and, for convenience, we assume this to be the case.

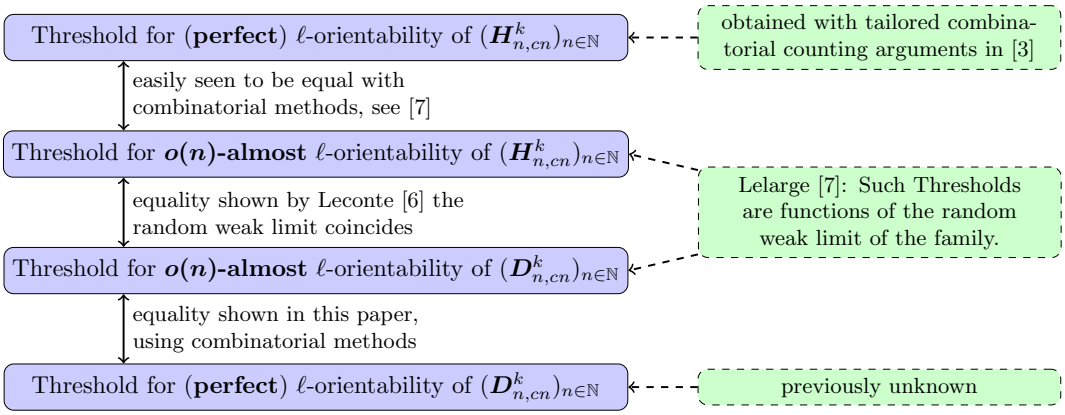


Figure 1 Two out of three steps for Theorem 1 are already known.

Orientability. A hypergraph $H = (V, E)$ is (perfectly) *ℓ-orientable* if there is function $f : E \to V$ mapping each edge $e \in E$ to an incident vertex $f(e) \in e$ such that each vertex v is the image of at most ℓ edges. For $d \in \mathbb{N}$ we say $H = (V, E)$ is *d-almost ℓ-orientable* if there is $E' \subseteq E$ of size $|E'| = |E| - d$ such that $H' = (V, E')$ is ℓ-orientable.

Orientability threshold. A family $(H_n)_{n\in\mathbb{N}}$ of random hypergraphs depending on a parameter c has an ℓ-orientability threshold $c^* \geq 0$ if for $c < c^*$, H_n is ℓ-orientable whp ("with high probability", i.e. with probability $1 - o_{n\to\infty}(1)$) and for $c > c^*$, H_n is *not* ℓ-orientable whp. Similarly we can define d-almost ℓ-orientability thresholds; we may even allow for d to be a function of n.

2 Outline of the Argument

Our goal is to prove the following theorem.

▶ **Theorem 1.** *For any fixed constants $k \geq 3, \ell \geq 1$, the ℓ-orientability threshold for $(H^k_{n,cn})_{n\in\mathbb{N}}$ and the ℓ-orientability threshold for $(D^k_{n,cn})_{n\in\mathbb{N}}$ coincide.*

We review what is known about double hashing in the context of cuckoo hashing, to explain what remains left to show (see Figure 1). Leconte [6] showed that the families $(H^k_{n,cn})_{n\in\mathbb{N}}$ and $(D^k_{n,cn})_{n\in\mathbb{N}}$ have the same Galton-Watson Tree as random weak limit. Lelarge [7] showed that the threshold for $o(n)$-almost ℓ-orientability of a graph family only depends on the random weak limit of the family. It is fairly easy to reconcile the ℓ-orientability and the $o(n)$-almost ℓ-orientability for $(H^k_{n,cn})_{n\in\mathbb{N}}$ (see [7]), showing that the thresholds are the same for that family. In order to establish Theorem 1 all we need to prove is an analogous result for $D^k_{n,cn}$, which is done in the following proposition. Note that ℓ-orientability trivially implies $o(n)$-almost ℓ-orientability, so only the non-trivial direction is given.

▶ **Proposition 1.** *Let $k \geq 3$ and $\ell \geq 1$ be fixed constants. Let c^* be the $o(n)$-almost ℓ-orientability threshold for $(D^k_{n,cn})_{n\in\mathbb{N}}$. Then for any $c < c^*$, $D^k_{n,cn}$ is ℓ-orientable whp.*

The proof uses two lemmas that are proved in Sections 3 and 4. To understand them, we need another concept. In the context of discussing ℓ-orientability of a hypergraph $H = (V, E)$, we call $V' \subseteq V$ a *Hall-witness* if the set $E(V')$ of edges induced by V' has size $|E(V')| > \ell \cdot |V'|$. By Hall's Theorem (restated in Section 4), H is ℓ-orientable if and only if no Hall-witness exists.

The lemmas we utilize are as follows:

▶ **Lemma 2.** *Let $k \geq 3$, $\ell \geq 1$ and $c > 0$ be fixed constants. Then there exists a constant $\delta > 0$ such that, whp, no Hall-witness of size less than δn exists for $D^k_{n,cn}$.*

▶ **Lemma 3.** *If $H = (V, E)$ is d-almost ℓ-orientable and $e \in E$ is contained in some minimal Hall-witness, then $H^{(e)} = (V, E - \{e\})$ is $(d-1)$-almost ℓ-orientable.*

Given these lemmas, we prove Proposition 1, following [7].

Proof of Proposition 1. Let $c = c^* - \varepsilon$ for some $\varepsilon > 0$. We may sample $D^k_{n,cn}$ by first sampling $D^k_{n,c'n}$ for $c' = c^* - \varepsilon/2$ and then removing $\varepsilon n/2$ edges. More precisely, we set $D^{(0)} := D^k_{n,c'n}$ and obtain $D^{(i+1)}$ from $D^{(i)}$ by removing an edge uniformly at random for $0 \leq i < \varepsilon n/2$. Then $D^{(\varepsilon n/2)}$ is distributed as $D^k_{n,cn}$.

For $0 \leq i \leq \varepsilon n/2$, let d_i be the smallest d such that $D^{(i)}$ is d-almost ℓ-orientable. By choice of c^* we have $d_0 = o(n)$ whp. We take δ from Lemma 2 and condition on the high probability event that any Hall-witness of $D^{(0)}$ has size at least δn. Of course, the same bound applies to Hall-witnesses of the subgraphs $D^{(i)}$ with $i > 1$.

Let i be an index with $d_i > 0$. Then $D^{(i)}$ is not ℓ-orientable and a minimal Hall-witness exists. Its size is at least δn, and it induces at least $\delta \ell n + 1$ edges. In particular, the probability that a random edge of $D^{(i)}$ is contained in this minimal Hall-witness is at least $\frac{\delta \ell n + 1}{c'n} \geq \delta \ell / c' = \Theta(1)$. If such an edge is chosen for removal, then by Lemma 3 we have $d_{i+1} = d_i - 1$. Until we reach $D^{(\varepsilon n/2)}$, there are $\varepsilon n/2 = \Theta(n)$ opportunities to reduce the d-value by 1, and each opportunity is realized with probability $\Theta(1)$. Since the initial gap is $o(n)$, the probability that we have $d_{\varepsilon n/2} > 0$ is $\Pr[X < o(n)]$ where $X \sim \mathrm{Bin}(\Theta(n), \Theta(1))$. Simple concentration bounds on binomial random variables prove that this is an $o(1)$-probability event, so we have $d_{\varepsilon n/2} = 0$ whp. Thus $D^{(\varepsilon n/2)} = D^k_{n,cn}$ is (perfectly) ℓ-orientable whp as desired. ◀

3 No small Hall-witness exists

In this section, we prove Lemma 2. We argue first that it is enough to prove the statement in the case $k = 3$ and $\ell = 1$. Indeed, if $D^k_{n,cn}$ contains no $V' \subseteq V$ inducing more than $|V'|$ edges, then certainly no such V' induces more than $\ell|V'|$ edges. Moreover, let us write $e = \{a_e + ib_e : 0 \leq i < k\}$ for an edge e of $D^k_{n,cn}$. We project each edge e in $D^k_{n,cn}$ to $e' = \{a_e + ib_e : 0 \leq i < 3\}$; then the resulting 3-uniform hypergraph is distributed like $D^3_{n,cn}$, and each $V' \subseteq V$ induces at least as many edges as in $D^k_{n,cn}$. It therefore suffices to show the unlikeliness of certain Hall witnesses in the case of $k = 3$, $\ell = 1$, and fixed $c \in \mathbb{R}^+$.

We introduce the notion of an (s,t)-*set*, a set of size s that contains precisely t arithmetic triples for some $3 \leq s \leq n$ and $1 \leq t \leq \binom{s}{2}$. More specifically, a subset S of $[0, n-1]$ contains some number of arithmetic triples modulo n, which unfortunately does not depend solely on the size of the subset S, and we therefore parametrize the number of triples with an additional variable t. Our plan is to use first moment methods and bound the sum:

$$\sum_{(s,t)} Q_{s,t} p_{s,t}$$

where $Q_{s,t}$ is the number of (s,t)-sets that could be minimal Hall-witnesses, and $p_{s,t}$ is an upper bound on the probability that an (s,t)-set actually is a minimal Hall-witness in $D^3_{n,cn}$.

We separately deal with the following ranges of the parameters s and t.

Case 1: Small s. If $s = o(n^{1/2})$ we exploit that $\sum_t Q_{s,t}$ is sufficiently small by direct counting.

Case 2: Medium s, small-ish t. For $s = \omega(n^{2/5})$ and $t \leq \frac{s^2}{4ce^2}$, the probability $t/\binom{n}{2} \approx \frac{2t}{n^2}$ that random edges are contained in such an (s,t)-set is small enough to find a good bound on $p_{s,t}$.

Case 3: Medium s, large t. For $\delta n \geq s = \omega(n^{2/5})$ (for a small δ chosen later) and $t > \frac{s^2}{4ce^2}$ it turns out that t far exceeds the number of arithmetic triples that would be expected from a random set of size s. A concentration bound by Warnke [14] then gives a useful bound on $Q_{s,t}$.

We deal with these three cases below. We use the following simple bounds on $p_{s,t}$. As we are working in the setting where $\ell = 1$, for a set of size s to be a minimal Hall-witness, there must be at least $s+1$ edges whose elements are in the set. We therefore find:

$$p_{s,t} \leq \left(\frac{t}{\binom{n}{2}}\right)^{s+1}\binom{cn}{s+1} \leq \left(\frac{2t}{n^2}\right)^{s+1}\left(\frac{cne}{s}\right)^{s+1} = \left(\frac{2cet}{sn}\right)^{s+1} \tag{1}$$

$$\leq \left(\frac{ces}{n}\right)^{s+1}. \tag{2}$$

The bound is derived by taking the probability that for a set of $s+1$ edges, each edge turns out to be one of the t arithmetic triples contained in S. This is multiplied with the number of ways to choose $s+1$ out of the cn edges of $D^3_{n,cn}$. For the second line we used the trivial bound $t \leq \binom{s}{2} \leq \frac{s^2}{2}$.

Case 1: $s = o(\sqrt{n})$. Assume $S \subseteq \mathbb{Z}_n$ is a minimal Hall-witness for $D^3_{n,cn}$ inducing a set P of edges (with $|P| > 3|S|$). As a hypergraph, (S,P) is spanning, i.e. each vertex is contained in an edge, otherwise the isolated vertex can be removed for a smaller Hall-witness. Also, (S,P) is connected, i.e. for any $x, y \in S$ there is a sequence $e_1, \dots, e_j \in P$ with $x \in e_1, y \in e_j$ and $e_i \cap e_{i+1} \neq \emptyset$ for $1 \leq i < j$). Otherwise, at least one connected component forms a smaller Hall-witness.

So for fixed s, we can count all (s,t)-sets (with arbitrary t) that might be minimal Hall-witnesses by counting vertex sets that can support connected spanning hypergraphs. We do this by counting *annotated depth-first-search-runs* (dfs-runs), associated with such (s,t)-sets, in the following way. A dfs-run through S starts at a root vertex $r \in S$ and puts it on the *stack*, whose topmost element is referred to as top. Then a sequence of steps follow, each of which either removes top from the stack (backtrack) or finds new vertices in S that are then put on the stack. More precisely, new vertices are found by specifying an arithmetic triple that is contained in S and involves top. The two vertices other than top may either both be new (find_2) or only one vertex is new, and a third vertex v was already found in a previous step (find_1). The following data about the dfs-run is needed to reconstruct S from it:

- The root vertex r. There are n possibilities.
- The type of each step, which can be backtrack, find_1 or find_2. Since there are at most $2s$ steps, there are at most 3^{2s} possibilities in total.
- For each step of type find_1, the vertex v that was previously found and that together with top and the new vertex forms an arithmetic triple. There are less than s possibilities. In addition we need the position of top and v in the arithmetic triple (essentially four possibilities). The newly discovered vertex can then be computed from top and v.

- For each step of type find_2, the difference between adjacent elements of the arithmetic triple – there are $n/2$ possibilities. Also, the position of top in that triple – there are 3 possibilities.

If f_1 and f_2 count the number of times the steps find_1 and find_2 are used in the dfs-run through S of size s, then we have $f_1 + 2f_2 = s - 1$. For $s = o(\sqrt{n})$, the find_2-steps yield a significantly higher number of possibilities per found vertex compared to find_1-steps, so we compute:

$$\sum_t Q_{s,t} \le n \cdot 3^{2s} \cdot (4s)^{f_1} (3n/2)^{f_2} \le n \cdot 3^{2s} \cdot (3n/2)^{(s-1)/2} \le c_1^s n^{(s+1)/2}$$

where c_1 is a constant. Using Equation (2) we get:

$$\sum_{s=3}^{o(\sqrt{n})} \sum_t Q_{s,t} p_{s,t} \le \sum_{s=3}^{o(\sqrt{n})} (\sum_t Q_{s,t})(\max_t p_{s,t}) \le \sum_{s=3}^{o(\sqrt{n})} c_1^s n^{(s+1)/2} \left(\frac{ces}{n}\right)^{s+1} \le \sum_{s=3}^{o(\sqrt{n})} \left(\frac{c_2 s}{\sqrt{n}}\right)^{s+1}$$

for a new constant c_2. Since each term in the sum is $O(n^{-2})$ and since there are $o(n^{1/2})$ terms, the sum is clearly $o(n^{-3/2}) = o(1)$, closing this case.

Case 2: $s = \omega(n^{2/5})$ and $t \le \frac{s^2}{4ce^2}$. Combining the trivial bound of $Q_{s,t} \le \binom{n}{s}$, Equation (1), and our assumption on t we obtain:

$$Q_{s,t} \cdot p_{s,t} \le \binom{n}{s} \cdot \left(\frac{2cet}{sn}\right)^{s+1} \le \left(\frac{ne}{s}\right)^s \cdot \left(\frac{s}{2ne}\right)^{s+1} \le \left(\frac{1}{2}\right)^s.$$

This is clearly $o(1)$, even after summing over all $O(n)$ admissible choices for s and all $O(n^2)$ choices for t.

Case 3: $\omega(n^{2/5}) \le s \le \delta n$ and $t > \frac{s^2}{4ce^2}$. A random set $S \subseteq \mathbb{Z}_n$ of size s in this range behaves very much like a random set T that is obtained by picking each element of $\mathbb{Z}_n$ independently with probability $p = \frac{s}{n}$. Let X be the number of arithmetic triples in T. We have $\mu := \mathbb{E}[X] = \binom{n}{2} p^3 \le \frac{s^3}{2n}$. In particular, the case $X > \frac{s^2}{4ce^2}$ is very rare if $s < \delta n$ for sufficiently small δ. We can therefore expect the number $Q_{s,t}$ to be significantly less than $\binom{n}{s}$. Formally we write:

$$Q_{s,t} \le \binom{n}{s} \Pr[S \text{ contains } t \text{ a.p.}] = \binom{n}{s} \Pr[X = t \mid |T| = s] \le \binom{n}{s} \Pr[X = t] O(\sqrt{n})$$

where $O(\sqrt{n})$ is the inverse of the probability of the event $|T| = s$. Using Theorem 1 from [14] with $k = 3$, $p = s/n$, we get positive constants $b, B > 0$ such that for sufficiently large n

$$\Pr[X = t] \le \Pr[X \ge (1 + \tfrac{t-\mu}{\mu})\mu] \overset{[14]}{\le} \mathrm{e}^{-b\sqrt{\frac{t-\mu}{\mu}}\sqrt{\mu}\log(1/p)}$$

Using our bound on t and assuming $\delta \le \frac{1}{4ce^2}$ we can bound the negated exponent by:

$$b\sqrt{t-\mu}\log(\tfrac{1}{p}) \ge b\sqrt{\tfrac{s^2}{4ce^2} - \tfrac{s^3}{2n}}\log(\tfrac{1}{\delta}) \ge bs\sqrt{\tfrac{1}{4ce^2} - \tfrac{\delta}{2}}\log(\tfrac{1}{\delta}) \ge bs\sqrt{\tfrac{1}{8ce^2}}\log(\tfrac{1}{\delta}) = sc_3\log(\tfrac{1}{\delta})$$

for some constant $c_3 > 0$ which yields $\Pr[X = t] \leq (\delta^{c_3})^s$. Combining this with Equation (2), this time assuming $\delta^{c_3} \leq \frac{1}{2ce^2}$, we can write

$$Q_{s,t} \cdot p_{s,t} \leq \binom{n}{s} \Pr[X = t] O(\sqrt{n}) \cdot p_{s,t} \leq \left(\frac{ne}{s}\right)^s (\delta^{c_3})^s O(\sqrt{n}) \left(\frac{ces}{n}\right)^{s+1}$$
$$\leq O(\sqrt{n}) \left(ce^2 \delta^{c_3}\right)^s \leq O(\sqrt{n}) \cdot 2^{-s}$$

which is $o(1)$, even when summing over all $\Theta(n)$ admissible values s and all $\Theta(n^2)$ admissible values for t.

4 The significance of Hall-witnesses

To understand how Hall's Theorem relates to our situation, we view a hypergraph $H = (V, E)$ as a bipartite graph with E on the "left", V on the "right" and a connection between $e \in E$ and $v \in V$ iff $v \in e$. We care about generalized $(1, \ell)$-matchings in this *incidence graph* of H, i.e. sets $M \subseteq E \times V$ such that any $e \in E$ has degree at most 1 in M and any $v \in V$ has degree at most ℓ in M. An ℓ-orientation f of H, viewed as a set of pairs $f \subseteq E \times V$, is then precisely an *edge-perfect* $(1, \ell)$-matching (each $e \in E$ has degree precisely 1). We call the corresponding notion of a vertex-perfect $(1, \ell)$-matching (each $v \in V$ has degree ℓ in M) an *ℓ-saturation*.

In this setting, Hall's Theorem is easily generalized to the following, where we use $N(X)$ to denote the direct neighbors of X in the incidence graph (note that $X \subseteq V$ and $X \subseteq E$ are both allowed) and $E(V')$ to denote the set of edges contained in $V' \subseteq V$.

▶ **Theorem 4** (Hall's Theorem).

(i) *H has an ℓ-orientation* $\Leftrightarrow \nexists E' \subseteq E$ *with* $\ell|N(E')| < |E'|$
$\Leftrightarrow \nexists V' \subseteq V$ *with* $\ell|V'| < |E(V')| \Leftrightarrow$: *No Hall-witness exists.*

(ii) *H has an ℓ-saturation* $\Leftrightarrow \nexists V' \subseteq V$ *with* $|N(V')| < \ell|V'|$.

We are now ready to prove Lemma 3.

Proof of Lemma 3. Let $H = (V, E)$ be a non-ℓ-orientable hypergraph and $S \subseteq V$ be a minimal Hall-witness to this fact. Consider $H_S = (S, E(S))$, the sub-hypergraph of H induced by S. Within H_S we have $|N_{H_S}(S')| > \ell|S'|$ for any $\emptyset \neq S' \subseteq S$, as otherwise, i.e. assuming $|N_{H_S}(S')| \leq \ell|S'|$, we have

$$|E(S - S')| = |E(S) - N_{H_S}(S')| = |E(S)| - |N_{H_S}(S')| > \ell|S| - \ell|S'| = \ell|S - S'|$$

which would make $S - S'$ a smaller Hall-witness than S, contradicting minimality.

This means for $H_S^{(e)} := (S, E(S) - \{e\})$ we have (replacing ">" with "≥") $|N_{H_S^{(e)}}(S')| \geq \ell|S'|$ for any $S' \subseteq S$ (the claim is trivial for $S' = \emptyset$). By Theorem 4(ii), $H_S^{(e)}$ has an ℓ-saturation $M_S^{(e)}$.

Now if H is d-almost ℓ-orientable and $M \subseteq E \times V$ is a corresponding $(1, \ell)$-matching of size $|E| - d$, our task is to obtain a $(1, \ell)$-matching M' with $|M| = |M'|$ in $H^{(e)} = (V, E - \{e\})$ where an edge $e \in E(S)$ was removed. This will imply that $H^{(e)}$ is $(d-1)$-almost ℓ-orientable as desired.

Constructing M' is easy, as we just remove all edges from $E(S)$ from M (this certainly gets rid of e if it was used) and re-saturate the vertices from S by adding an appropriate subset $Y \subseteq M_S^{(e)}$. Then $M' := (M \setminus E(S)) \cup Y$ has the same size as M. ◀

5 Conclusion

We have shown that for ℓ-orientations in k-ary cuckoo hashing (for constant k and ℓ), double hashing yields the same load thresholds as fully random hashing. This provides yet another example of a hashing structure with the same behavior when using only double hashing in place of random hashing. It seems somewhat unfortunate and perhaps a little mysterious that there does not yet appear to be a unifying argument for multiple such hashing structures; each structure, thus far, has required its own specialized argument. We optimistically suggest that a more unified approach may exist, that would shed more light on this phenomenon.

A problem closely related to the cuckoo hashing problem we have studied here is the question of the ℓ-core threshold of a k-uniform random hypergraph. The ℓ-core of a hypergraph is obtained by repeatedly removing any vertex of degree less than ℓ, and all adjacent edges. One can think of the ℓ-core as what is left after a "greedy" first stage in an offline algorithm for finding an $(\ell-1)$-orientation; each bucket with at most $\ell-1$ objects simply accepts those objects, and the remaining objects would then have to be more carefully placed to obtain an an $(\ell-1)$-orientation, if possible. For random hypergraphs on n vertices with cn edges, there are similar thresholds $c^* = c^*(k,\ell)$ for the existence of a non-empty ℓ-core; that is for $c < c^*$ the ℓ-core is empty with high probability, and for $c > c^*$ the ℓ-core consists of $\Omega(n)$ edges with high probability. Empirically, the double-hashing graph appears to have the same thresholds as random hypergraphs for the ℓ-core, and it is known the thresholds are the same when $\ell > k$ [12]. It might seem our approach would be useful for settling this question as well, but thus far we cannot currently rule out small $o(n)$-sized ℓ-cores under double hashing using these ideas. This question remains tantalizingly open.

References

1. Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.
2. Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.
3. Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The Multiple-Orientability Thresholds for Random Hypergraphs. In *Proc. 22nd SODA*, pages 1222–1236, 2011. URL: `http://www.siam.org/proceedings/soda/2011/SODA11_092_fountoulakisn.pdf`.
4. Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better bloom filter. *Random Structures & Algorithms*, 33(2):187–218, 2008.
5. M. Leconte, M. Lelarge, and L. Massoulié. Convergence of multivariate belief propagation, with applications to cuckoo hashing and load balancing. In *Proc. 24th SODA*, pages 35–46, 2013. URL: `http://dl.acm.org/citation.cfm?id=2627817.2627820`.
6. Mathieu Leconte. Double hashing thresholds via local weak convergence. In *51st Annual Allerton Conference on Communication, Control, and Computing, Allerton 2013, Allerton Park & Retreat Center, Monticello, IL, USA, October 2-4, 2013*, pages 131–137. IEEE, 2013. `doi:10.1109/Allerton.2013.6736515`.
7. Marc Lelarge. A New Approach to the Orientation of Random Hypergraphs. In *Proc. 23rd SODA*, pages 251–264, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095139`.
8. George S Lueker and Mariko Molodowitch. More analysis of double hashing. *Combinatorica*, 13(1):83–96, 1993.
9. Michael Mitzenmacher. Some open questions related to cuckoo hashing. In *ESA*, pages 1–10. Springer, 2009.

10 Michael Mitzenmacher. Balanced allocations and double hashing. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 331–342. ACM, 2014.

11 Michael Mitzenmacher. More analysis of double hashing for balanced allocations. *CoRR*, abs/1503.00658, 2015. `arXiv:1503.00658`.

12 Michael Mitzenmacher and Justin Thaler. Peeling arguments and double hashing. In *50th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2012, Allerton Park & Retreat Center, Monticello, IL, USA, October 1-5, 2012*, pages 1118–1125. IEEE, 2012. `doi:10.1109/Allerton.2012.6483344`.

13 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. `doi:10.1016/j.jalgor.2003.12.002`.

14 Lutz Warnke. Upper tails for arithmetic progressions in random subsets. *Israel Journal of Mathematics*, pages 1–49, 7 2017. `doi:10.1007/s11856-017-1546-3`.

A Greedy Algorithm for Subspace Approximation Problem

Nguyen Kim Thang
IBISC, Univ Evry, University Paris Saclay
Evry, France
thang@ibisc.fr
https://orcid.org/0000-0002-6085-9453

Abstract

In the subspace approximation problem, given m points in $\mathbb{R}^n$ and an integer $k \leq n$, the goal is to find a k-dimension subspace of $\mathbb{R}^n$ that minimizes the ℓ_p-norm of the Euclidean distances to the given points. This problem generalizes several subspace approximation problems and has applications from statistics, machine learning, signal processing to biology. Deshpande et al. [4] gave a randomized $O(\sqrt{p})$-approximation and this bound is proved to be tight assuming NP $\neq$ P by Guruswami et al. [7]. It is an intriguing question of determining the performance guarantee of deterministic algorithms for the problem. In this paper, we present a simple deterministic $O(\sqrt{p})$-approximation algorithm with also a simple analysis. That definitely settles the status of the problem in term of approximation up to a constant factor. Besides, the simplicity of the algorithm makes it practically appealing.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Subspace Approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.30

Funding Research supported by the ANR project OATA n° ANR-15-CE40-0015-01.

1 Introduction

Massive data in high dimension emerge naturally in many domains from machine learning to biology. It has been observed that although data lie in high-dimensional spaces, in practice they have low intrinsic dimension. Dimension-reduction algorithms are essential in many domains such as image processing, personalized medicine, etc. In this paper, we consider the following subspace approximation problem in the context of capturing the underlying low-dimensional structures of given data.

Subspace Problem. Given points $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{R}^n$ and integers $p \geq 1$ and $0 \leq k \leq n$. Find a k-dimensional linear subspace W that minimizes the ℓ_p-norms of Euclidean distances of these points to W, i.e.,

$$\min_{W:\dim(W)=k} \left(\sum_{i=1}^{m} d(\boldsymbol{a}_i, W)^p \right)^{1/p}$$

The Subspace Problem, which is introduced by Deshpande et al. [4], is a generalization of several sub-space approximation problems which have been widely studied. For example, the well-known Least Square Fit Problem is a particular case. In the latter, given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $0 \leq k \leq n$, find a matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ of rank at most k that minimizes the Frobenius norm of the difference $\|\boldsymbol{A} - \boldsymbol{B}\|_F := \left(\sum_{i,j} (A_{ij} - B_{ij})^2\right)^{1/2}$. Taking the rows of $\boldsymbol{A}$

to be $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ and $p = 2$, Subspace Problem reduces to Least Square Fit Problem. Another special case of Subspace Problem is Radii Problem. In the latter, given points $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{R}^n$, their *outer* $(n-k)$ *radius* is the minimum, over all k-dimensional linear subspaces, of the maximum Euclidean distances of these points to the subspace. This problem is equivalent to Subspace Problem with $p = \infty$. Moreover, Subspace Problem is related to other problems such as the L_p-Grothendieck Problem [8] and ℓ_p-Regression Problem [1, 5, 3]. We refer the reader to [4] for more details about the connection between these problems.

Deshpande et al. [4] introduced the Subspace Problem and gave a randomized $O(\sqrt{p})$-approximation algorithm. In their approach, they consider a convex relaxation that optimizes over positive semidefinite matrices and the rank constraint is replaced by a trace constraint. Subsequently, the solution $\boldsymbol{X}$ of the convex relaxation is rounded to a matrix of suitable rank. Intuitively, the authors divide singular vectors of the solution $\boldsymbol{X}$ into several bins and constructs one vector for each bin by taking a Bernoulli random linear combination of vectors within each bin. The analysis is carried out by powerful techniques coupled with properties of the p^{th}-moment of sums of Bernoulli random variables. Besides, Deshpande et al. [4] proved that the Subspace Problem is hard to approximate within a factor $\Omega(\sqrt{p})$ assuming the Unique Games Conjecture (UGC). Later on, bypassing the need for UGC, Guruswami et al. [7] showed that the problem is indeed NP-hard to approximate within a factor $\Omega(\sqrt{p})$.

Our Contribution. In this paper, we present a deterministic greedy algorithm with the same $O(\sqrt{p})$-approximation guarantee. Informally, at any step, the algorithm greedily extends the subspace in order to minimizes the marginal cost of the objective functions. The algorithm (and also the analysis) is extremely simple, which makes it practically appealing. Besides, our algorithm is deterministic whereas the one in [4] is randomized. The analysis is based on a smooth inequality (Lemma 2), which has been originally used in the context of algorithmic game theory in order to bound the quality of equilibrium (price of anarchy) in scheduling games [2]. This allows us to give a deterministic algorithm instead of the randomized one [4] which crucially relies on concentration inequalities in functional analysis in order to bound the moments of sums of Bernoulli random variables. Our result definitely settles the status of the problem in term of approximation up to a constant factor.

Related works. The most closely related to our paper is [4] where the results have been summarized earlier. For the Least Square Fit Problem, the optimal subspace is spanned by the top k right singular vector of $\boldsymbol{A}$ and that can be computed in time $O(\min\{n^2m, nm^2\})$ [6]. For the Radii Problem, $O(\sqrt{\log m})$-approximation with $k = n-1$ can be implied from the works of Nesterov [10] and Nemirovski et al. [9]. Later on, Varadarajan et al. [11] gave an $O(\sqrt{\log m})$-approximation algorithm for arbitrary k. Note that it is well-known that ℓ_∞-norm can be approximated by $\ell_{\log m}$-norm up to a constant factor. Hence, $O(\sqrt{\log m})$-approximation can be deduced from [4] (so our work) by choosing $p = \log m$.

2 Greedy Algorithm

As in [4], we use a formulation of the Subspace Problem in terms of the orthogonal complement of the subspace W. Specifically, let $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-k}$ be an orthonormal basis for the orthogonal complement. Let $\boldsymbol{Z} \in \mathbb{R}^{n \times (n-k)}$ be the matrix with the j^{th} column vector $\boldsymbol{z}_j$. Then $d(\boldsymbol{a}_i, W) = \|\boldsymbol{a}_i^T \boldsymbol{Z}\|_2$. Hence, the problem is to find an orthonormal basis $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-k}$ of a

$(n-k)$-dim vector space V so that the corresponding matrix $\boldsymbol{Z}$ minimizes

$$\sum_{i=1}^{m} \|\boldsymbol{a}_i^T \boldsymbol{Z}\|_2^p = \sum_{i=1}^{m} \left(\sum_{\ell=1}^{n-k} (\boldsymbol{a}_i^T \boldsymbol{z}_\ell)^2 \right)^{p/2}$$
$$= \sum_{i=1}^{m} \sum_{j=1}^{n-k} \left[\left(\sum_{\ell=1}^{j} (\boldsymbol{a}_i^T \boldsymbol{z}_\ell)^2 \right)^{p/2} - \left(\sum_{\ell=1}^{j-1} (\boldsymbol{a}_i^T \boldsymbol{z}_\ell)^2 \right)^{p/2} \right],$$

where conventionally the sum with no term equals 0.

Algorithm. Initially, the subspace $U_0 = \emptyset$. For $1 \le j \le n-k$, choose a vector $\boldsymbol{u}_j \neq \boldsymbol{0}$ orthonormal to the subspace U_{j-1} spanned by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{j-1}$ such that it minimizes the marginal increase of the objective, i.e.,

$$\boldsymbol{u}_j \in \arg\min_{\boldsymbol{z} \perp U_{j-1}} \sum_{i=1}^{m} \left[\left((\boldsymbol{a}_i^T \boldsymbol{z})^2 + \sum_{\ell=1}^{j-1} (\boldsymbol{a}_i^T \boldsymbol{u}_\ell)^2 \right)^{p/2} - \left(\sum_{\ell=1}^{j-1} (\boldsymbol{a}_i^T \boldsymbol{u}_\ell)^2 \right)^{p/2} \right]$$

In fact, vector $\boldsymbol{u}_j$ can be computed by solving a convex program. Specifically, let $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{n-j+1}\}$ be an arbitrary orthogonal basis of the vector space $U_{j-1}^{\perp}$ (which is orthogonal to U_{j-1}). Computing $\boldsymbol{u}_j$ is equivalent to computing the coefficients $b_1, \ldots, b_{n-j+1}$ in the decomposition of $\boldsymbol{u}_j$ in the basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{n-j+1}\}$. The convex program is

$$\min_{b_1, \ldots, b_{n-j+1}} \sum_{i=1}^{m} \left[\left((\boldsymbol{a}_i^T \cdot \sum_{h=1}^{n-j+1} b_h \boldsymbol{e}_h)^2 + \sum_{\ell=1}^{j-1} (\boldsymbol{a}_i^T \boldsymbol{u}_\ell)^2 \right)^{p/2} - \left(\sum_{\ell=1}^{j-1} (\boldsymbol{a}_i^T \boldsymbol{u}_\ell)^2 \right)^{p/2} \right]$$
$$\sum_{h=1}^{n-j+1} b_h^2 = 1$$
$$b_1, \ldots, b_{n-j+1} \in \mathbb{R}$$

Note that in the convex program, variables are $b_1, \ldots, b_{n-j+1}$ (the vectors $\boldsymbol{u}_\ell$'s, $\boldsymbol{e}_h$'s have been already determined).

Analysis. Let $\boldsymbol{V}$ be an optimal $n \times (n-k)$ matrix and V be the corresponding vector space spanned by column vectors of $\boldsymbol{V}$. In the remaining, we will show that

$$\left(\sum_{i=1}^{m} \|\boldsymbol{a}_i^T \boldsymbol{U}\|_2^p \right)^{1/p} \le O(\gamma_p) \cdot \left(\sum_{i=1}^{m} \|\boldsymbol{a}_i^T \boldsymbol{V}\|_2^p \right)^{1/p}$$

First, we recall the following standard lemma.

▶ **Lemma 1.** *For any $(n-k)$-dim subspace V, there exists an orthonormal basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-k}\}$ of V such that $\boldsymbol{v}_j$ is orthogonal to U_{j-1} for $1 \le j \le n-k$.*

Proof. We construct an orthogonal basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-k}\}$ of V by induction. The orthonormal basis is obtained by standard normalizing procedure. For $j = 1$, any arbitrary vector $\boldsymbol{v}_1 \in V$ is perpendicular to U_0. Assume that vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$ for $j < (n-k)$ have been constructed so that they satisfy the lemma. Since the subspace V has dimension $(n-k)$ which is strictly larger than j, there exits a vector $\boldsymbol{w}_{j+1} \in V$ which is independent to $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$. Define vector $\boldsymbol{v}_{j+1} := \boldsymbol{w}_{j+1} - \mathsf{Pr}_{\boldsymbol{u}_j}(\boldsymbol{w}_{j+1})$ where $\mathsf{Pr}_{\boldsymbol{u}_j}(\boldsymbol{w}_{j+1})$ is the projection of vector $\boldsymbol{w}_{j+1}$ onto the subspace U_j. So $\boldsymbol{v}_{j+1}$ is orthogonal to $\boldsymbol{u}_j$ and is independent to $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$. ◀

▶ **Lemma 2.** *For any given vector $\boldsymbol{a}$, for arbitrary vectors $\boldsymbol{u}_\ell$ and $\boldsymbol{v}_\ell$ with $1 \le \ell \le n-k$, it holds that*

$$\sum_{j=1}^{n-k}\left[\left((a^T\boldsymbol{v}_j)^2+\sum_{\ell=1}^{j-1}\left(a^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}-\left(\sum_{\ell=1}^{j-1}\left(a^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}\right] \le \mu\left(\sum_{\ell=1}^{n-k}\left(a^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}+\lambda\left(\sum_{\ell=1}^{n-k}\left(a^T\boldsymbol{v}_\ell\right)^2\right)^{p/2}$$

where $\lambda = O\big((\alpha\cdot\frac{p}{2})^{\frac{p}{2}-1}\big)$ for some constant α and $\mu=\frac{p/2-1}{p/2}$.

Proof. Denote $b_j=(a^T\boldsymbol{v}_j)^2$ and $c_j=(a^T\boldsymbol{u}_j)^2$ for $1\le j\le n-k$. The lemma inequality reads

$$\sum_{j=1}^{n-k}\left[\left(b_j+\sum_{\ell=1}^{j-1}c_\ell\right)^{p/2}-\left(\sum_{\ell=1}^{j-1}c_\ell\right)^{p/2}\right]\le\mu\left(\sum_{\ell=1}^{n-k}c_\ell\right)^{p/2}+\lambda\left(\sum_{\ell=1}^{n-k}b_\ell\right)^{p/2}$$

The inequality holds for $\lambda = O\big((\alpha\cdot\frac{p}{2})^{\frac{p}{2}-1}\big)$ for some constant α and $\mu=\frac{p/2-1}{p/2}$, which has been proved in [2] in the context of algorithmic game theory. For completeness, we put the proof of the above inequality in the appendix (Lemma 5). ◀

▶ **Theorem 3.** *The greedy algorithm is $O(\sqrt{p})$-approximation.*

Proof. Recall that $\boldsymbol{U}$ be the solution of the algorithm where the j^{th} column vector is $\boldsymbol{u}_j$ for $1\le j\le n-k$. Let $\boldsymbol{v}_1,\ldots,\boldsymbol{v}_{n-k}$ be an orthonormal basis of V that satisfies Lemma 1. We have

$$\begin{aligned}
\sum_{i=1}^{m}\|\boldsymbol{a}_i^T\boldsymbol{U}\|_2^p &= \sum_{i=1}^{m}\sum_{j=1}^{n-k}\left[\left(\sum_{\ell=1}^{j}\left(\boldsymbol{a}_i^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}-\left(\sum_{\ell=1}^{j-1}\left(\boldsymbol{a}_i^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}\right]\\
&\le \sum_{i=1}^{m}\sum_{j=1}^{n-k}\left[\left((\boldsymbol{a}_i^T\boldsymbol{v}_j)^2+\sum_{\ell=1}^{j-1}\left(\boldsymbol{a}_i^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}-\left(\sum_{\ell=1}^{j-1}\left(\boldsymbol{a}_i^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}\right]\\
&\le \sum_{i=1}^{m}\left[\mu\left(\sum_{\ell=1}^{n-k}\left(\boldsymbol{a}_i^T\boldsymbol{u}_\ell\right)^2\right)^{p/2}+\lambda\left(\sum_{\ell=1}^{n-k}\left(\boldsymbol{a}_i^T\boldsymbol{v}_\ell\right)^2\right)^{p/2}\right]\\
&= \mu\sum_{i=1}^{m}\|\boldsymbol{a}_i^T\boldsymbol{U}\|_2^p+\lambda\sum_{i=1}^{m}\|\boldsymbol{a}_i^T\boldsymbol{V}\|_2^p
\end{aligned}$$

The first inequality is due to the choice of the algorithm at any step j (note that $\boldsymbol{v}_j\perp U_{j-1}$ so $\boldsymbol{v}_j$ is a candidate at step j). The second inequality holds by Lemma 2 where $\lambda = O\big((\alpha\cdot\frac{p}{2})^{\frac{p}{2}-1}\big)$ for some constant α and $\mu=\frac{p/2-1}{p/2}$. Rearranging the terms and taking the p^{th}-root, we get

$$\left(\sum_{i=1}^{m}\|\boldsymbol{a}_i^T\boldsymbol{U}\|_2^p\right)^{1/p}\le O(\sqrt{p})\cdot\left(\sum_{i=1}^{m}\|\boldsymbol{a}_i^T\boldsymbol{V}\|_2^p\right)^{1/p}$$

◀

References

1 Kenneth L Clarkson. Subgradient and sampling algorithms for ℓ_1 regression. In *Proc. 16th Symposium on Discrete algorithms*, pages 257–266, 2005.

2 Johanne Cohen, Christoph Dürr, and Nguyen Kim Thang. Smooth inequalities and equilibrium inefficiency in scheduling games. In *Internet and Network Economics*, pages 350–363, 2012.

3 Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W Mahoney. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

4 Amit Deshpande, Madhur Tulsiani, and Nisheeth K Vishnoi. Algorithms and hardness for subspace approximation. In *Proc. 22nd Symposium on Discrete Algorithms*, pages 482–496, 2011.

5 Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Sampling algorithms for ℓ_2 regression and applications. In *Proc. 17th Symposium on Discrete algorithm*, pages 1127–1136, 2006.

6 Gene H Golub and Charles F Van Loan. *Matrix computations*. Johns Hopkins University Press, 2012.

7 Venkatesan Guruswami, Prasad Raghavendra, Rishi Saket, and Yi Wu. Bypassing UGC from some optimal geometric inapproximability results. *ACM Transactions on Algorithms*, 12(1):6, 2016.

8 Guy Kindler, Assaf Naor, and Gideon Schechtman. The UGC hardness threshold of the L_p grothendieck problem. *Mathematics of Operations Research*, 35(2):267–283, 2010.

9 Arkadi Nemirovski, Cornelis Roos, and Tamás Terlaky. On maximization of quadratic form over intersection of ellipsoids with common center. *Mathematical Programming*, 86(3):463–473, 1999.

10 Yuri Nesterov. *Global quadratic optimization via conic relaxation*. Université Catholique de Louvain. Center for Operations Research and Econometrics [CORE], 1998.

11 Kasturi Varadarajan, Srinivasan Venkatesh, Yinyu Ye, and Jiawei Zhang. Approximating the radii of point sets. *SIAM Journal on Computing*, 36(6):1764–1776, 2007.

Appendix: Technical Lemmas

In this section, we show technical lemmas. The following lemma has been proved in [2]. We give it here for completeness.

▶ **Lemma 4** ([2]). *Let k be a positive integer. Let $0 < a(k) \leq 1$ be a function on k. Then, for any $x, y > 0$, it holds that*

$$y(x+y)^k \leq \frac{k}{k+1} a(k) x^{k+1} + b(k) y^{k+1}$$

where α is some constant and

$$b(k) = \begin{cases} \Theta\left(\alpha^k \cdot \left(\frac{k}{\log k a(k)}\right)^{k-1}\right) & \text{if } \lim_{k\to\infty}(k-1)a(k) = \infty, \quad (1a) \\ \Theta\left(\alpha^k \cdot k^{k-1}\right) & \text{if } (k-1)a(k) \text{ are bounded } \forall k, \quad (1b) \\ \Theta\left(\alpha^k \cdot \frac{1}{k a(k)^k}\right) & \text{if } \lim_{k\to\infty}(k-1)a(k) = 0. \quad (1c) \end{cases}$$

Proof. Let $f(z) := \frac{k}{k+1} a(k) z^{k+1} - (1+z)^k + b(k)$. To show the claim, it is equivalent to prove that $f(z) \geq 0$ for all $z > 0$.

We have $f'(z) = ka(k)z^k - k(1+z)^{k-1}$. We claim that the equation $f'(z) = 0$ has an unique positive root z_0. Consider the equation $f'(z) = 0$ for $z > 0$. It is equivalent to

$$\left(\frac{1}{z} + 1\right)^k \cdot \frac{1}{z} = a(k)$$

The left-hand side is a strictly decreasing function and the limits when z tends to 0 and ∞ are ∞ and 0, respectively. As $a(k)$ is a positive constant, there exists an unique root $z_0 > 0$.

Observe that function f is decreasing in $(0, z_0)$ and increasing in $(z_0, +\infty)$, so $f(z) \geq f(z_0)$ for all $z > 0$. Hence, by choosing

$$b(k) = \left|\frac{k}{k+1}a(k)z_0^{k+1} - (1+z_0)^k\right| = (1+z_0)^{k-1}\left(1 + \frac{z_0}{k+1}\right) \tag{2}$$

it follows that $f(z) \geq 0\ \forall z > 0$.

We study the positive root z_0 of equation

$$a(k)z^k - (1+z)^{k-1} = 0 \tag{3}$$

Note that $f'(1) = k(a(k) - 2^{k-1}) < 0$ since $0 < a(k) \leq 1$. Thus, $z_0 > 1$. For the sake of simplicity, we define the function $g(k)$ such that $z_0 = \frac{k-1}{g(k)}$ where $0 < g(k) < k-1$. Equation (3) is equivalent to

$$\left(1 + \frac{g(k)}{k-1}\right)^{k-1} g(k) = (k-1)a(k)$$

Note that $e^{w/2} < 1 + w < e^w$ for $w \in (0, 1)$. For $w := \frac{g(k)}{k-1}$, we obtain the following upper and lower bounds for the term $(k-1)a(k)$:

$$e^{g(k)/2}g(k) < (k-1)a(k) < e^{g(k)}g(k) \tag{4}$$

Recall the definition of *Lambert W function*. For each $y \in \mathbb{R}^+$, $W(y)$ is defined to be solution of the equation $xe^x = y$. Note that, xe^x is increasing with respect to x, hence $W(\cdot)$ is increasing.

By definition of the Lambert W function and Equation (4), we get that

$$W\left((k-1)a(k)\right) < g(k) < 2W\left(\frac{(k-1)a(k)}{2}\right) \tag{5}$$

First, consider the case where $\lim_{k\to\infty}(k-1)a(k) = \infty$. The asymptotic sequence for $W(x)$ as $x \to +\infty$ is the following: $W(x) = \ln x - \ln\ln x + \frac{\ln\ln x}{\ln x} + O\left(\left(\frac{\ln\ln x}{\ln x}\right)^2\right)$. So, for large enough k, $W((k-1)a(k)) = \Theta(\log((k-1)a(k)))$. Since $z_0 = \frac{k-1}{g(k)}$, from Equation (5), we get $z_0 = \Theta\left(\frac{k}{\log(ka(k))}\right)$. Therefore, by (2) we have $b(k) = \Theta\left(\alpha^k \cdot \left(\frac{k}{\log ka(k)}\right)^{k-1}\right)$ for some constant α.

Second, consider the case where $(k-1)a(k)$ is bounded by some constants. So by (5), we have $g(k) = \Theta(1)$. Therefore $z_0 = \Theta(k)$ which again implies $b(k) = \Theta\left(\alpha^k \cdot k^{k-1}\right)$ for some constant α.

Third, we consider the case where $\lim_{k\to\infty}(k-1)a(k) = 0$. We focus on the Taylor series W_0 of W around 0. It can be found using the Lagrange inversion and is given by

$$W_0(x) = \sum_{i=1}^{\infty} \frac{(-i)^{i-1}}{i!}x^i = x - x^2 + O(1)x^3.$$

Thus, for k large enough $g(k) = \Theta((k-1)a(k))$. Hence, $z_0 = \Theta(1/a(k))$. Once again that implies $b(k) = \Theta\left(\alpha^k \cdot \frac{1}{ka(k)^k}\right)$ for some constant α. ◀

▶ **Lemma 5.** *For any sequences of non-negative real numbers* $\{a_1, a_2, \ldots, a_n\}$ *and* $\{b_1, b_2, \ldots, b_n\}$ *and for any polynomial* g *of degree* k *with non-negative coefficients, it holds that*

$$\sum_{i=1}^{n}\left[g\left(b_i + \sum_{j=1}^{i-1} a_j\right) - g\left(\sum_{j=1}^{i-1} a_j\right)\right] \leq \lambda(k) \cdot g\left(\sum_{i=1}^{n} b_i\right) + \mu(k) \cdot g\left(\sum_{i=1}^{n} a_i\right)$$

where $\mu(k) = \frac{k-1}{k}$ *and* $\lambda(k) = \Theta\left(k^{k-1}\right)$. *The same inequality holds for* $\mu(k) = \frac{k-1}{k \ln k}$ *and* $\lambda(k) = \Theta\left((\alpha \cdot k \ln k)^{k-1}\right)$ *for some constant* α.

Proof. Let $g(z) = g_0 z^k + g_1 z^{k-1} + \cdot + g_k$ with $g_t \geq 0\ \forall t$. The lemma holds since it holds for every z^t for $0 \leq t \leq k$. Specifically,

$$\begin{aligned}
\sum_{i=1}^{n}\left[g\left(b_i + \sum_{j=1}^{i-1} a_j\right) - g\left(\sum_{j=1}^{i-1} a_j\right)\right] &= \sum_{t=1}^{k} g_{k-t} \cdot \sum_{i=1}^{n}\left[\left(b_i + \sum_{j=1}^{i-1} a_j\right)^t - \left(\sum_{j=1}^{i-1} a_j\right)^t\right] \\
&\leq \sum_{t=1}^{k} g_{k-t} \cdot \left[t \cdot b_i \cdot \left(b_i + \sum_{j=1}^{i-1} a_j\right)^{t-1}\right] \leq \sum_{t=1}^{k} g_{k-t} \cdot \left[\lambda(t)\left(\sum_{i=1}^{n} b_i\right)^t + \mu(t)\left(\sum_{i=1}^{n} a_i\right)^t\right] \\
&\leq \lambda(k) \cdot g\left(\sum_{i=1}^{n} b_i\right) + \mu(k) \cdot g\left(\sum_{i=1}^{n} a_i\right)
\end{aligned}$$

The first inequality follows the convex inequality $(x+y)^{k+1} - x^{k+1} \leq (k+1)y(x+y)^k$. The second inequality follows Lemma 4 (Case (1b) and $a(k) = 1/(k+1)$). The last inequality holds since $\mu(t) \leq \mu(k)$ and $\lambda(t) \leq \lambda(k)$ for $t \leq k$. ◀

Planar 3-SAT with a Clause/Variable Cycle

Alexander Pilz[1]
Department of Computer Science, ETH Zürich. Zürich, Switzerland
alexander.pilz@inf.ethz.ch
https://orcid.org/0000-0002-6059-1821

Abstract

In the Planar 3-SAT problem, we are given a 3-SAT formula together with its incidence graph, which is planar, and are asked whether this formula is satisfiable. Since Lichtenstein's proof that this problem is NP-complete, it has been used as a starting point for a large number of reductions. In the course of this research, different restrictions on the incidence graph of the formula have been devised, for which the problem also remains hard.

In this paper, we investigate the restriction in which we require that the incidence graph is augmented by the edges of a Hamiltonian cycle that first passes through all variables and then through all clauses, in a way that the resulting graph is still planar. We show that the problem of deciding satisfiability of a 3-SAT formula remains NP-complete even if the incidence graph is restricted in that way and the Hamiltonian cycle is given. This complements previous results demanding cycles only through either the variables or clauses.

The problem remains hard for monotone formulas and instances with exactly three distinct variables per clause. In the course of this investigation, we show that monotone instances of Planar 3-SAT with three distinct variables per clause are always satisfiable, thus settling the question by Darmann, Döcker, and Dorn on the complexity of this problem variant in a surprising way.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases 3-SAT, 1-in-3-SAT, planar graph

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.31

Related Version `https://arxiv.org/abs/1710.07476`

Acknowledgements The Linked Planar 3-SAT problem has been brought to our attention by Lena Schlipf. We thank Erik Demaine for pointing out the motivation in platform games, Patrick Schnider and Michael Hoffmann for valuable discussions, as well as an anonymous reviewer for the indicated simplification.

1 Introduction

Let ϕ be a Boolean formula in conjunctive normal form (CNF) and let G_ϕ be a graph whose vertices are the variables and the clauses of ϕ such that (1) every edge of G_ϕ is between a variable and a clause and (2) there is an edge between a variable v and a clause c if and only if v occurs in c (negated or unnegated). We call ϕ a *CNF formula* and G_ϕ is called the *incidence graph* of ϕ. A CNF formula is a *3-SAT formula* if every clause contains at most three variables. (We will also discuss the case where every clause contains exactly three distinct variables.) The Planar 3-SAT problem asks whether a given 3-SAT formula ϕ is

[1] Supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35.

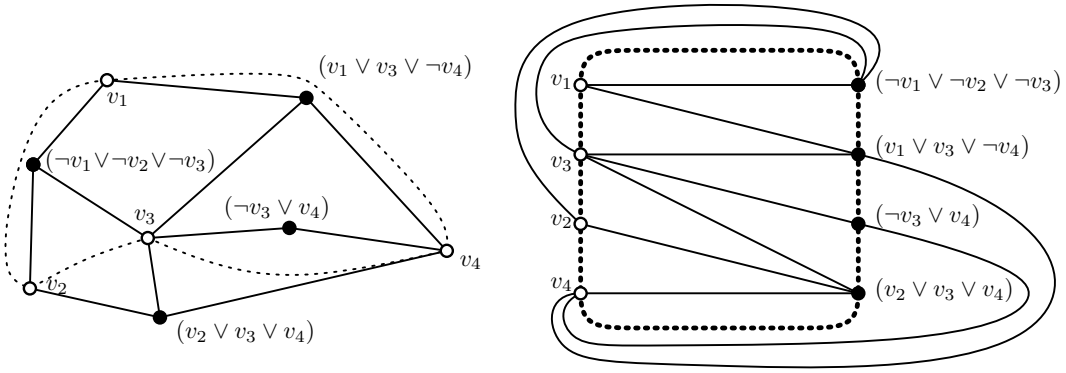


Figure 1 Left: An incidence graph for a 3-SAT formula that is planar. Clauses are black vertices and variables are white vertices. The graph is augmented by a spanning cycle (dotted) on the variable vertices. Right: A Linked Planar 3-SAT instance, with a Hamiltonian cycle κ (dotted) passing through the variables and the clauses. (At most two edges of κ may already be present in the incidence graph.)

satisfiable, given that G_ϕ is a planar graph. This problem has been shown to be NP-complete by Lichtenstein [21]. (In contrast to the general version, a PTAS is known for maximizing the number of satisfied clauses for the planar version of the 3-SAT problem [16].) See Figure 1 for drawings of an incidence graph.

Reducing from Planar 3-SAT is a standard technique to show NP-hardness of problems in computational geometry. In these reductions, the vertices and edges of G_ϕ are replaced by gadgets (consisting of geometric objects) that influence each other. However, it is often useful to have further restrictions on G_ϕ or on how G_ϕ can be embedded.

Lichtenstein's reduction already contains such a restriction: the problem remains NP-complete even if the graph remains planar after adding a cycle whose vertices are exactly the variables of the formula [21], and this cycle is part of the input. We call it a *variable cycle*. This fact allows for placing the variable vertices along a line, connected to "three-legged" clauses above and below that line (stated more explicitly in [18]; see Figure 2). De Berg and Khosravi [7] showed that it is also possible to have all literals in the clauses above the line to be positive, and all literals in clauses below the line negative. In Lichtenstein's reduction, one may as well add a *clause cycle* whose vertices are exactly the clauses of the formula while keeping the graph planar [20]. We call the Planar 3-SAT variants in which we require a variable cycle and a clause cycle Var-Linked Planar 3-SAT and Clause-Linked Planar 3-SAT, respectively.[2] Even though the incidence graphs constructed by Lichtenstein can be augmented with both a clause cycle [20] and a variable cycle, one cannot adapt Lichtenstein's construction to always obtain both cycles without any crossings.[3]

Our research is motivated by the following problem that attempts to combine these restrictions. See Figure 1 (right) for an accompanying illustration.

▶ **Definition 1** (Linked Planar 3-SAT)**.** Let $G_\phi = (C \cup V, E)$ be the incidence graph of a 3-SAT formula ϕ, where C is the set of clauses and V is the set of variables of ϕ. Further,

[2] While Lichtenstein's definition of planar 3-SAT [21] already requires the cycle through the variables, this property is often considered an explicit restriction. We thus follow the terminology of Fellows et al. [11] to emphasize when the variable cycle is needed.

[3] If not stated otherwise, we will implicitly require the incidence graphs augmented by the additional edges to be planar throughout this paper.

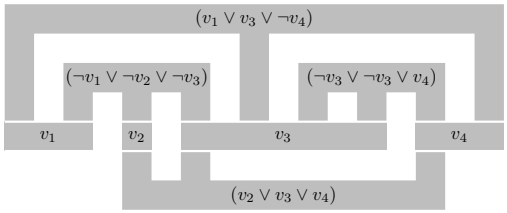


Figure 2 A "three-legged" Planar 3-SAT instance with variables on a line similar to [18, p. 425]. (There, two-variable clauses are transformed to three-variable clauses that contain one literal twice, a construction that is not necessary but possible for the initial graph in our reduction.)

let κ be a Hamiltonian cycle of $C \cup V$ that first visits all elements of C and then all elements of V. Suppose that the union of G_ϕ and κ is a planar graph. The Linked Planar 3-SAT problem asks, given ϕ, G_ϕ, and κ, whether ϕ is satisfiable.

Related problems in which all variables or all clauses can be drawn incident to the unbounded face are known to be in P, due to results by Knuth [17], and Kratochvíl and Křivánek [19], respectively. In particular, this is the case when there is a variable cycle and a path connecting all clauses or vice versa. One way of tackling the Linked Planar 3-SAT problem could be to show that G_ϕ has bounded treewidth. For such instances, the satisfiability of ϕ can be decided in polynomial time [12]. (This generalizes the above-mentioned results, as every k-outerplanar graph has treewidth at most $3k - 1$ [4]; see also Demaine's lecture notes [10].) However, with the right perspective on the Linked Planar 3-SAT problem, it will be easy to observe that there are formulas whose incidence graph has a grid minor with a linear number of vertices (and thus such graphs have unbounded treewidth). It is the same perspective through which we will show NP-completeness of the problem in Section 2, using a reduction from Planar 3-SAT. We note that requiring an arbitrary Hamiltonian cycle is not a restriction: As the incidence graph is bipartite, it is known that its page number is two [8]; hence, we can always add a Hamiltonian cycle through the variables and clauses in a planar way (possibly re-using edges of the incidence graph).

1.1 Motivation

Restrictions on the problem to reduce from can make NP-hardness reductions simpler. For reductions from Planar 3-SAT, it is common to actually reduce from Var-Linked Planar 3-SAT, using the variable cycle, in particular the "three-legged" embedding of [18]. Also, the clause cycle has been used [20, 11]. For an exhaustive survey on the numerous variants of Planar 3-SAT, see the thesis of Tippenhauer [27].

One motivation for considering Linked Planar 3-SAT is the framework for showing NP-hardness of platform games by Aloupis et al. [2]. In this class of reductions from 3-SAT to such games, a player's character starts at a specified position and traverses all variable gadgets, making a decision on their truth value. Clauses connected to the satisfied literal can be "unlocked" by visiting these clauses. Finally the player's character has to traverse all clause gadgets to reach the finish (called the "check path"). The framework then requires a game-specific implementation of the gadgets for start, finish, variables, clauses, and crossovers. Reducing from Linked Planar 3-SAT removes this dependency on crossover gadgets. (In particular, Theorem 9 can be used to show that the traversal through the literals can be done without crossings.) See [2, Section 2.1] for a more detailed description.

1.2 Results

We first prove that Linked Planar 3-SAT is NP-complete. In the following sections, we refine the construction to show that restricted variants of the problem remain hard as well. In particular, we do this for formulas without negated and unnegated variables in the same clause (Monotone Planar 3-SAT), and formulas for which the edges to negated variables are all on the same side of κ (recall that κ is the Hamiltonian cycle that first visits all variables and then all clauses). Also, we may require that all clauses contain exactly three distinct variables. A cycle κ in instances of Positive Planar 1-in-3-SAT (which requires exactly one true literal in each clause) also keeps the problem hard.

Finally, we discuss settings in which the planarity constraint is fulfilled only by satisfiable formulas. In particular, we show that planar CNF formulas with at least four variables per clause are always satisfiable. The same holds for instances of Monotone Planar 3-SAT with exactly three variables per clause. This solves an open problem by Darmann, Döcker, and Dorn [5, 6], who show that the corresponding problem with at most three variables per clause remains NP-complete with bounds on the variable occurrences, which refines the result of de Berg and Khosravi [7].

2 NP-hardness of Linked Planar 3-SAT

We now show that Linked Planar 3-SAT is NP-hard by reducing Planar 3-SAT to it. We are thus given a 3-SAT formula $\tilde{\phi}$ with variable set $\tilde{V}$, clause set $\tilde{C}$, and incidence graph $G_{\tilde{\phi}}$. Our goal is to construct another formula ϕ that is an instance of Linked Planar 3-SAT with incidence graph G_ϕ and Hamiltonian cycle κ such that ϕ is satisfiable if and only if $\tilde{\phi}$ is. The construction of G_ϕ will be given by an embedding.

We start by producing a suitable embedding of the initial graph $G_{\tilde{\phi}}$. We require an embedding Γ of $G_{\tilde{\phi}}$ on the integer grid, with edges drawn as x-monotone curves. (Our construction can easily be modified for non-x-monotone edges, but this assumption facilitates the presentation.) We further require that the size of the grid is polynomial in the size of $\tilde{\phi}$. It is well-known that a planar graph with n vertices can be embedded with straight-line edges on a $O(n) \times O(n)$ grid in $O(n)$ time [9, 26]. We can take such an embedding and perturb the clause vertices s.t. each variable has even x-coordinate, and the x-coordinate of each clause is odd. This can be done without introducing crossings by choosing the grid sufficiently large and scaling Γ; a blow-up by a factor polynomial in n is sufficient.[4] More specifically, we scale Γ by a polynomial multiple of 2 and increase the x-coordinate of each clause vertex by 1. (It will become apparent that our reduction merely uses the combinatorial structure of the embedding; however, it seems easier to describe the construction with a fixed straight-line embedding of $G_{\tilde{\phi}}$ on the integer grid.)

With a suitable drawing of $G_{\tilde{\phi}}$ at hand, we start the drawing of G_ϕ with the curve that will contain the cycle κ (we add its vertices later). It can be partitioned into two paths, one that will contain the elements of the variable set V (κ_V) and one for the elements of the clause set (κ_C). In our drawing shown in Figure 3 (left), we obtain a rectangular region R, whose intersection with κ consists of vertical line segments of unit distance, in alternation belonging to κ_V and κ_C. We call them the *clause segments* and *variable segments*, respectively. We

[4] We can consider the smallest horizontal distance between a vertex and a non-incident edge. This distance v is rational with numerator and denominator quadratic in the largest coordinate. By multiplying the x-coordinates of the vertices by $2/v$, rounding, and again multiplying by 2, we get the desired embedding, similar to [1, Lemma 6.1].

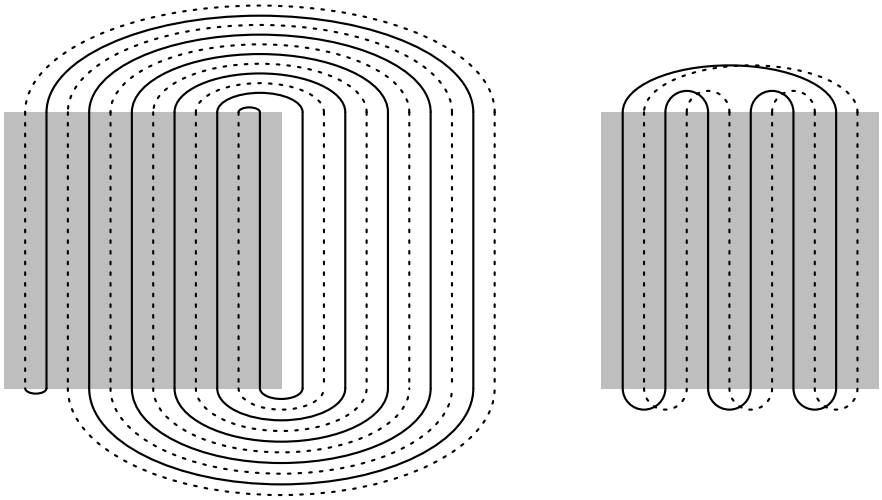


Figure 3 Left: Drawings of the paths κ_C (solid) and κ_V (dotted), containing the clauses and variables of ϕ, respectively. They intersect a rectangle R (gray) in vertical segments. The incidence graph is drawn inside R. Right: A similar construction can be used to show that all the incidence graphs we obtain can be augmented by either a clause cycle or a variable cycle (and not just two paths obtained from κ).

assume that the segments are placed on the integer grid with unit distance, with the variable segments having even x-coordinates and the clause segments having odd x-coordinates. In other words, R represents a grid in which the columns are traversed in alternation by κ_V and κ_C.

Place Γ inside R. By the construction of γ, all its variable vertices have even x-coordinates and are thus on variable segments, and its clause vertices are on clause segments. We obtain G_ϕ by replacing the edges of Γ by gadgets, consisting of subgraphs of G_ϕ. In the construction, we will make use of "cyclic implications", effectively copying the value of a variable; the graph G_ϕ will contain many pairs of variables x and x' with a clause $c_x = (\neg x \vee x')$ and a clause $c'_x = (x \vee \neg x')$. Clearly, $x = x'$ in any satisfying assignment. We depict the negation in such a clause c_x by an arrow from x to c_x, and from c_x to x' (where the arrows are also edges of the incidence graph). In general, we use the convention that an arrow from a variable to a clause denotes that the variable occurs negated in that clause, while an arrow from the clause to the variable means that the variable occurs unnegated.

We replace each edge e of Γ by a sequence of so-called *connector gadgets*. A connector gadget consists of two variables x and x', and two clauses $c_x = (\neg x \vee x')$ and $c'_x = (\neg x' \vee x)$, implying $x = x'$ in any satisfying truth assignment. The variable vertices are placed on the intersections of e with two consecutive variable segments in R, and the clause vertices are placed on the clause segment between them (also close to the crossing of e and the clause segment), effectively subdividing κ. An edge in Γ connecting a variable $\tilde{v} \in \tilde{V}$ to a clause $\tilde{c} \in \tilde{C}$ that crosses κ can be replaced by a sequence of connector gadgets in a sufficiently small neighborhood of the edge, as shown in Figure 4. Thus, in the resulting drawing, we have subdivided κ to remove crossings with e, and the resulting formula is satisfiable if and only if the initial formula is satisfiable.

Replacing the edges of Γ by the connector gadgets results in a drawing of G_ϕ. As all edges are x-monotone, all crossings with κ are replaced. Thus this drawing is planar and contains a number of vertices that is polynomial in $|C|$, as the number of crossings of an edge

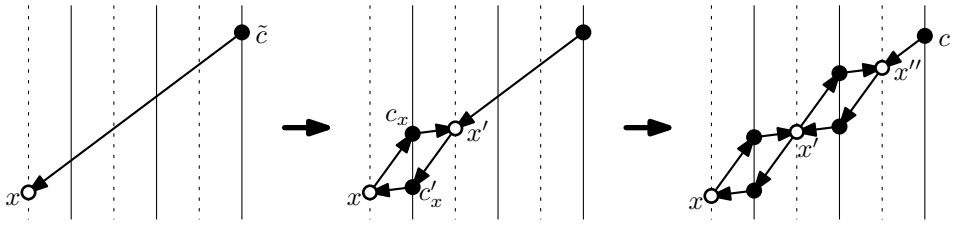


Figure 4 An edge of the initial incidence graph crossing the cycle κ (left) can be replaced by a sequence of connector gadgets. Variable vertices are white dots, clause vertices are black. The two variables in the connector gadget (middle) have the same value, and the occurrence of a variable x in the clause $\tilde{c}$ is replaced by the last variable x'' to obtain an equivalent clause c.

in Γ with clause and variable segments (and thus the number of vertices needed to replace the edge) is bounded by the grid size. Also, all "new" clauses contain only two variables (we will see a modification of the reduction without this property). As none of the edges in the resulting embedding of G_ϕ crosses the initially drawn cycle for κ, G_ϕ can be augmented by κ along that cycle maintaining planarity. Finally, observe that ϕ is satisfiable if and only if $\tilde{\phi}$ is: the two variables of a connector gadget must have the same value, and the clauses not part of the connector gadget are the clauses of $\tilde{C}$ in which we replaced variables by others that have to be equal. We thus obtain our main result.

▶ **Theorem 2.** Linked Planar 3-SAT *is NP-complete.*

Observe that our construction also does not change the number of satisfiable assignments, i.e., the reduction is *parsimonious*. Since counting the number of satisfiable assignments to a planar 3-SAT formula is #P-complete [15], this also holds for Linked Planar 3-SAT.

3 Further variants

We use the main idea of drawing the cycle κ as shown in Figure 3 to obtain similar reductions for variants of the planar satisfiability problem. If the initial problem is known to be hard, we merely need to find an according connector gadget to replace the crossings of edges of $G_{\tilde{\phi}}$ and κ.

3.1 Positive Planar 1-in-3-SAT

It is easy to transform our reduction to be an instance of 1-in-3-SAT, where exactly one variable is true: we get the "main" clauses directly from an initial 1-in-3-SAT instance, and the connector gadgets are the same. We therefore impose two more requirements on the instance. First, all clauses should have three elements, and second, all literals should be positive.

▶ **Definition 3** (Positive Planar 1-in-3-SAT [24])**.** Given a formula ϕ in which each clause contains exactly three distinct unnegated literals, and an embedding of the incidence graph G_ϕ, the Positive Planar 1-in-3-SAT problem asks whether there exists a satisfying assignment of ϕ such that exactly one variable in each clause is true.

Mulzer and Rote [24] show that deciding satisfiability of Positive Planar 1-in-3-SAT instances is hard, even if the incidence graph can be augmented by a variable cycle.

▶ **Theorem 4.** *The* Positive Planar 1-in-3-SAT *problem remains NP-complete even for problem instances that are also instances of the* Linked Planar 3-SAT *problem.*

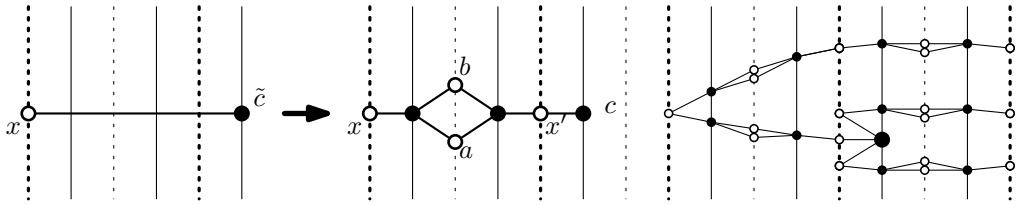


Figure 5 Reduction for the Positive Planar 1-in-3-SAT variant. If $x = 1$, then both $a = 0$ and $b = 0$, and thus $x' = 1$ as well. Otherwise, one of a and b is 1 and we again have $x = x'$. Since the connector gadgets have width 4, we may "overshoot" when coming from the right.

Proof. The reduction works again by blowing up the grid embedding of the initial Positive Planar 1-in-3-SAT incidence graph by a polynomial factor f and increase the x-coordinate of each clause by 1. We choose f to be a multiple of 8, as (i) all variable vertices should have even x-coordinates (and are thus placed on the appropriate part of κ), (ii) the number of clause segments between a variable and a clause (before increasing its x-coordinate by 1) is a multiple of two (because of the gadget width described below), and (iii) there are at least three variable segments between each clause vertex of the initial instance and each of its variables, even after increasing the x-coordinate of each clause vertex by 1 (to have sufficient space for "detours" described below). As before, f is chosen such that moving the clauses does not produce any crossings in the straight-line embedding. Then, we use two clauses sharing two variables to produce a connector gadget, as shown in Figure 5. The two clauses (x, a, b) and (x', a, b) ensure that $x = x'$.

The construction works analogous to the simpler connector gadget of the previous section,[5] except for the following caveat. The connector gadgets have width 4, and this is the reason for requiring condition (ii): We placed each clause of the initial formula at positions with x-coordinate $8k + 1$, for some integer k. The sketch on the right of Figure 5 shows how to connect the clause to the connector gadgets. (Note the "detour" one chain of connector gadgets may have to take in the vicinity of the initial clause to connect to it if all three edges of the clause emanate to the right in the initial drawing; this does not interfere with other gadgets as f is a multiple of 8.) ◀

3.2 Exactly three distinct variables per clause

If we require the formula to have exactly three distinct variables in each clause, the reduction can also be modified accordingly. Mansfield [22] showed how to extend Lichtenstein's construction to obtain a planar 3-SAT formula with exactly three different variables per clause[6] by constructing such a formula with planar incidence graph and a variable that is false in every satisfying assignment.

▶ **Theorem 5.** *The* Linked Planar 3-SAT *problem remains NP-complete even if each clause contains exactly three different variables.*

Proof. We reduce from the non-linked version of the problem by drawing an incidence graph on our grid and replace edge parts by gadgets that transport the truth settings of a variable. To this end, we use a modified connector gadget shown in Figure 6, which assure that two

[5] We thank an anonymous referee for simplifying the previously-used gadget.
[6] This restriction has been re-discovered, e.g., in [15].

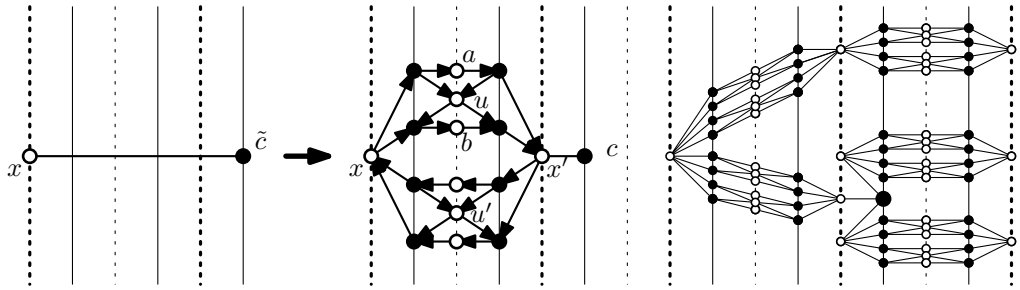


Figure 6 A modified connector gadget. The leftmost variable has to have the same value as the rightmost one. All clauses have exactly three distinct variables. An arrow from a variable to a clause indicates that the variable occurs negated. For an arrow from the clause to the variable, the occurrence is unnegated.

variables x and x' always have the same value: When removing u from the shown formula, we get $x \Rightarrow a$ and $a \Rightarrow x'$, as well as $x \Rightarrow b$ and $b \Rightarrow x'$. Since u occurs unnegated in the clauses containing a, and it occurs negated in the clauses containing b, one of the two implication pairs will make sure that $x \Rightarrow x'$. More formally, we have $(\neg x \vee a \vee u) \wedge (\neg a \vee u \vee x')$, which entails $(\neg x \vee u \vee x')$, and $(\neg x \vee b \vee \neg u) \wedge (\neg b \vee \neg u \vee x')$, which entails $(\neg x \vee \neg u \vee x')$. These two clauses have $(\neg x \vee x')$ as a resolvent. Analogously, the clauses containing u' imply $x' \Rightarrow x$.

We note that in Mansfield's reduction it is no longer shown that there exists a variable cycle, so we cannot rely on a "three-legged" embedding but rather use a straight-line one. As in the previous reduction, the connector gadgets have width 4. We thus again use a suitable blow-up factor and place each clause of the initial formula at positions with x-coordinate $8k + 1$, for some integer k. ◀

Note that the bi-implication implemented by the connector gadget has sixteen different truth assignments, independent of whether x is true or false. As the Planar 3-SAT problem is known to be #P-complete even if each clause contains exactly three distinct variables [15], we can add connector gadgets to transform any such instance into a Linked Planar 3-SAT instance. For each connector gadget, the number of solutions is multiplied by 16.

▶ **Theorem 6.** *The* Linked Planar 3-SAT *problem remains #P-complete even if each clause contains exactly three different variables.*

3.3 Monotonicity restrictions

We can add the following list of restrictions to the setting for which the problem remains hard. Both can be shown by reducing from Monotone Planar 3-SAT.

A clause is *monotone* if it contains either only negated or only unnegated literals; a formula is *monotone* if all its clauses are monotone.

▶ **Definition 7** (Monotone Planar 3-SAT)**.** Let $G_\phi = (C \cup V, E)$ be the incidence graph of a 3-SAT formula ϕ, where C is the set of clauses and V is the set of variables of ϕ, and each clause of C is monotone. The Monotone Planar 3-SAT problem asks whether ϕ is satisfiable.

This problem has been shown to be NP-complete by de Berg and Khosravi [7]; they actually show that the problem remains hard even if there is a variable cycle separating the clauses with the negated variables from the ones with the unnegated variables. That is,

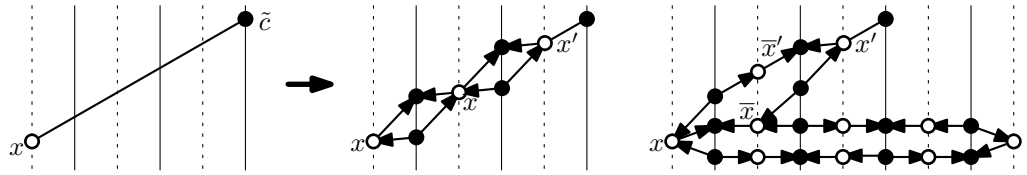


Figure 7 A variant of the connector gadget in which all clauses are monotone. We have $x \neq \overline{x} \neq x' = x$ in the middle. The gadget can be further split to have each variable vertex of degree at most three (right).

the incidence graph can be drawn in a rectilinear way, with the variables on the x-axis and exactly the clauses with the negated occurrences below the x-axis. We can use their result to show hardness of the according problem in our setting.

▶ **Theorem 8.** *The* Linked Planar 3-SAT *problem remains NP-complete even if all clauses are monotone.*

Proof. We reduce from Monotone Planar 3-SAT. The reduction is the same as for general Linked Planar 3-SAT, but with a different connector gadget. We use two subgadgets that contain two clauses, one all positive one all negative, that ensure that two variables are not equal. Two such gadgets in sequence form a new connector gadget. See Figure 7. We can deal with the position of the clause being off by one to the end of each connector gadget in the same way as in Figure 6 (right). ◀

Lichtenstein already showed that Planar 3-SAT remains NP-complete even if we require that the variable cycle partitions the edges of every variable vertex into those leading to a negated occurrence and those leading to an unnegated one [21, Lemma 1]. (As he mentions, this also implies that one could split a variable vertex into two literal vertices while preserving planarity.) Note that there is a subtle difference to the restriction of de Berg and Khosravi [7] to Monotone Planar 3-SAT, as the side of the variable cycle to which the edges to the negated occurrences emanate is not fixed globally. The following set of related restrictions could be particularly interesting for further reductions.

▶ **Theorem 9.** *The* Linked Planar 3-SAT *problem remains NP-complete even if, for each clause, the edges corresponding to positive occurrences emanate to the interior of the cycle κ, and the ones to negated occurrences to the exterior. In addition, each variable occurs in at most three clauses.*

Proof. We reduce from the Monotone Planar 3-SAT variant of de Berg and Khosravi [7], using a "three-legged" embedding. The construction (see Figure 8) uses cycles consisting of the variables $x_1, \dots x_k$ and $\overline{x_1}, \dots, \overline{x_k}$ and clauses $(\neg x_i \vee x_{i+1})$ and $(\neg \overline{x_i} \vee \overline{x_{i+1}})$ for all valid indices, as well as the clauses $(x_1 \vee \overline{x_1})$ and $(\neg x_k \vee \neg \overline{x_k})$. The latter thus entails $(\neg x_1 \vee \neg \overline{x_1})$. We therefore have $x_i = x_j$, $\overline{x_i} = \overline{x_j}$, and $x_i \neq \overline{x_j}$ for $i, j \leq k$. The variable vertices are placed from left to right with increasing indices. Thus, the variables on the upper part of the cycle have the opposite value of those on the lower part. For a clause with positive literals in the Monotone Planar 3-SAT instance, we connect the variables as in Figure 8.

We therefore have variable gadgets that consist of said cycles; see the three bottom-most cycles in Figure 8. These variable gadgets are connected to the clause c (which represents the clause of the initial formula) either directly (for the middle variable gadget), or via another cycle (for the left and the right variable gadget). For these connector gadgets, we again have that the variables on the upper part have the opposite value as those of the lower part; in the

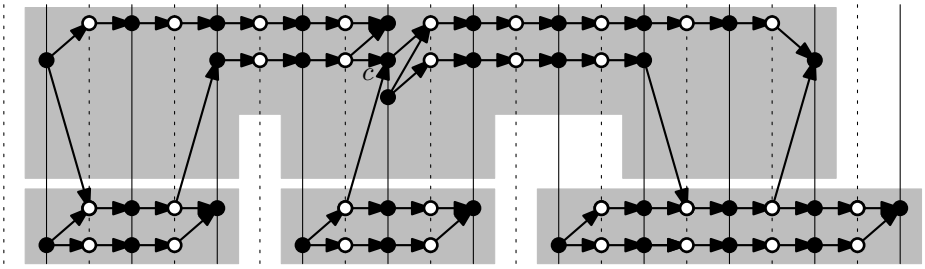


Figure 8 All clauses have negative literals to the left and positive literals to the right. The variable state is transported by cycles where the variables on top have the negative value of the variables at the bottom. A clause $\tilde{c} = (x \vee y \vee z)$ is transformed to $c = (\neg\overline{x_i} \vee \neg\overline{y_j} \vee z_k)$. The "three-legged" embedding of the clause is indicated by the gray contour.

way a variable gadget and a connector gadget are connected, a variable on the upper part of the variable gadget has the same value as one on the lower part of the connector gadget. The clause c has two edges to the left (which therefore correspond to negated occurrences) and one to the right (an unnegated occurrence). In the setting shown in Figure 8), we consider a variable set to true if the variables on the lower part of the variable gadget are true. A clause $\tilde{c} = (x \vee y \vee z)$ is thus transformed to $c = (\neg\overline{x_i} \vee \neg\overline{y_j} \vee z_k)$. We can have variable gadgets of arbitrary width simply by having larger cycles, and connect the clauses according to the "three-legged" embedding. The resulting construction is thus crossing-free, and the formula is satisfiable if and only if the initial one is. ◀

Observe that the connector gadgets in Figure 8 use only clauses with two variables. The reduction therefore also works for Planar 2-SAT instances. While Planar 2-SAT can be solved in polynomial time, it is known to be #P-complete [29]. As our gadgets are parsimonious (i.e., do not change the number of solutions), they can be applied to show #P-completeness.

▶ **Corollary 10.** *The* Linked Planar 2-SAT *problem is #P-complete. It remains #P-complete even if for each clause, the edges corresponding to positive occurrences emanate to the interior of the cycle κ, and the ones to negative literals to the exterior.*

Note that making each variable occur in at most three clauses requires that there are clauses with at most two (different) literals, as every CNF formula with exactly three literals per clause and at most three occurrences per variable is satisfiable [28].

Darmann, Döcker, and Dorn [5, 6] showed how to reduce the number of times a variable occurs. For the variant of Monotone Planar 3-SAT that requires exactly three different variables per clause, the complexity was previously unknown [5, 6]. Surprisingly, it turns out that such instances are always satisfiable, as discussed in the next section.

4 Remark: different cycles through clauses and variables

Observe that, for all our constructions, G_ϕ still allows for adding a variable cycle H, as well as a clause cycle H', as shown in Figure 3 (right). But these cycles will, in general, cross mutually. Also, they will cross the cycle κ. Recall that if H did not cross H' or κ_C, the problem would be solvable in polynomial time [10].

Using the gadgets and the two cycles shown in Figure 3 (right), we observe that Planar 3-SAT remains NP-complete even if these cycles exist for all variants mentioned. While the variable and clause cycles have been identified in [21] and [20], respectively, it seems to

Figure 9 Clause (black) with at least three distinct variables (white). We can augment the graph with a 3-cycle through variables (dashed) and remove the clauses while preserving planarity. In any 4-coloring of the graph, the three variables belong to three different color classes and thus there always exists a satisfying truth assignment by setting the variables of two color classes to true.

have been unknown for the variants using exactly three variables per clause (even though Mansfield's construction [22] can be embedded to obtain the cycles). Also, it seems that clause cycles have not been considered for Monotone Planar 3-SAT and Planar Positive 1-in-3-SAT.

5 Properties forcing satisfiability

Planarity is a rather drastic combinatorial restriction on the structure of a graph. While NP-completeness of 3-SAT is preserved in the planar setting, further properties may lead not only to polynomial-time algorithms (as for Planar NAE-SAT [23]), but also to instances that are always satisfiable.

▶ **Theorem 11.** *Every instance of* Planar SAT *in which each clause has three negated or three unnegated occurrences of three distinct variables is satisfiable. A satisfying assignment can be found in quadratic time.*

Proof. Consider any plane drawing of the incidence graph of the formula. For each clause vertex, we can add a 3-cycle consisting of three of its variable vertices that are either all negated or all unnegated, as shown in Figure 9. (For clauses with three variables this is similar to a Y-Δ transform, a common operation to replace a vertex of degree 3 by a 3-cycle: connect two variables by a curve in a neighborhood of the two edges connecting them to the clause.) Observe that, after removing the clause vertices, the resulting graph is still planar. It is thus 4-colorable [3] and we may consider any 4-coloring of the variables. Set the variables of the vertices with colors 1 and 2 to true, and the others to false. A 3-cycle contains three different colors, and thus a 3-cycle through three (monotone) variable vertices has at least one variable set to true and one variable set to false. A 4-coloring can be found in quadratic time [25]. ◀

▶ **Corollary 12.** *Every instance of* Monotone Planar SAT *with at least three distinct variables per clause is satisfiable.*

▶ **Corollary 13.** *Every instance of* Planar SAT *with at least five distinct variables per clause is satisfiable.*

For at least four variables per clause, we can give a similar result, closing the gap between Corollary 13 and Mansfield's result [22], using less heavy machinery than the Four-Color theorem. The proof makes use of Hall's theorem [13], inspired by a technique by Tovey [28]: each subset of k clauses has at least k variables occurring in it.

▶ **Lemma 14.** *Let $G = (B \cup C, E)$ be a bipartite planar graph with parts B and C such that all vertices in C have degree at least four. Then there exists a matching covering every vertex of C.*

Proof. Consider any plane embedding of G and let F be its set of faces. Thus, by Euler's formula, we have $|B| + |C| - |E| + |F| = 1 + k$, where $k \geq 1$ is the number of connected components of G. Every edge has two incidences with faces, and since G is bipartite, every face has at least four incidences with edges. We get $2|E| = \sum_{f \in F} |f| \geq 4|F|$ (where $|f|$ is the number of sides of the face f). Combining this with Euler's formula to $|E| \geq 2(1 + k - |B| - |C| + |E|)$ gives the bound

$$2|B| + 2|C| - 2 - 2k \geq |E| \ . \tag{1}$$

Further, $|E| \geq 4|C|$, as G is bipartite and every element of C is incident to at least four edges. Combining this with (1) results in

$$|C| \leq |B| - 1 - k \ . \tag{2}$$

Finally, observe that every subset of C plus its neighbors in B also induce a graph in which the analogue of (2) holds. As, for every subset of C, the set of adjacent elements in B has at least the same cardinality, there is a matching on G that covers C by Hall's theorem [13]. ◀

▶ **Theorem 15.** *Each CNF formula with planar incidence graph of n vertices and at least four distinct variables per clause has a satisfying assignment, which can be found in $O(n^{1.5})$ time.*

Proof. Let ϕ be a planar SAT instance with vertex set V and clause set C, and let $G_\phi = (V \cup C, E)$ be the associated incidence graph. By Lemma 14, there is a matching on G_ϕ covering C, which assigns a distinct variable to each clause (i.e., a system of distinct representatives for the clauses). If we set the according literal to true, we get an assignment satisfying ϕ. The matching can be found using the algorithm by Hopcroft and Karp [14]. ◀

References

1. Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is np-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015. doi:10.1007/s00454-015-9709-7.
2. Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic nintendo games are (computationally) hard. *Theor. Comput. Sci.*, 586:135–160, 2015. doi:10.1016/j.tcs.2015.02.037.
3. Kenneth Appel and Wolfgang Haken. *Every Planar Map is Four-Colorable.* Number 98 in Contemporary Mathematics. AMS, 1989. With the collaboration of J. Koch.
4. Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
5. Andreas Darmann, Janosch Döcker, and Britta Dorn. On planar variants of the monotone satisfiability problem with bounded variable appearances. *CoRR*, abs/1604.05588, 2016. arXiv:1604.05588.
6. Andreas Darmann, Janosch Döcker, and Britta Dorn. The monotone satisfiability problem with bounded variable appearances. *Int. J. Foundations Comp. Sci.*, 2017. To appear.
7. Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *Int. J. Comput. Geometry Appl.*, 22(3):187–206, 2012. URL: http://www.worldscinet.com/doi/abs/10.1142/S0218195912500045.
8. Hubert de Fraysseix, Patrice Ossona de Mendez, and János Pach. A left-first search algorithm for planar graphs. *Discrete & Computational Geometry*, 13:459–468, 1995. doi:10.1007/BF02574056.

9 Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. doi:10.1007/BF02122694.

10 Erik Demaine. Algorithmic lower bounds: Fun with hardness proofs; Lecture 7. http://courses.csail.mit.edu/6.890/fall14/scribe/lec7.pdf. Scribe: Quanquan Liu, Jeffrey Bosboom. Retrieved June 21, 2017.

11 Michael R. Fellows, Jan Kratochvíl, Matthias Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995. doi:10.1007/BF01190507.

12 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. doi:10.1016/j.dam.2006.06.020.

13 Philip Hall. On representatives of subsets. *J. London Math. Soc.*, s1-10(1):26–30, 1935.

14 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

15 Harry B. Hunt III, Madhav V. Marathe, Venkatesh Radhakrishnan, and Richard Edwin Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167, 1998. doi:10.1137/S0097539793304601.

16 Sanjeev Khanna and Rajeev Motwani. Towards a syntactic characterization of PTAS. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 329–337. ACM, 1996. doi:10.1145/237814.237979.

17 Donald E. Knuth. Nested satisfiability. *Acta Inf.*, 28(1):1–6, 1990. doi:10.1007/BF02983372.

18 Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Discret. Math.*, 5(3):422–427, 1992.

19 Jan Kratochvíl and Mirko Krivánek. Satisfiability of co-nested formulas. *Acta Inf.*, 30(4):397–403, 1993. doi:10.1007/BF01209713.

20 Jan Kratochvíl, Anna Lubiw, and Jaroslav Nesetril. Noncrossing subgraphs in topological layouts. *SIAM J. Discrete Math.*, 4(2):223–244, 1991. doi:10.1137/0404022.

21 David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:329–343, 1982.

22 Anthony Mansfield. Determining the thickness of graphs is NP-hard. *Math. Proc. Cambridge Phil. Soc.*, 93(01):9–23, 1983.

23 Bernard M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, 1988.

24 Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is np-hard. *J. ACM*, 55(2):11:1–11:29, 2008. doi:10.1145/1346330.1346336.

25 Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 571–575. ACM, 1996. doi:10.1145/237814.238005.

26 Walter Schnyder. Embedding planar graphs on the grid. In David S. Johnson, editor, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California.*, pages 138–148. SIAM, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320191.

27 Simon Tippenhauer. On planar 3-SAT and its variants. Master's thesis, Freie Universität Berlin, 2016.

28 Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

29 Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.

Tree-Residue Vertex-Breaking: a new tool for proving hardness

Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA.
edemaine@mit.edu

Mikhail Rudoy
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA. Now at Google Inc.
mrudoy@gmail.com
https://orcid.org/0000-0002-9210-1006

Abstract

In this paper, we introduce a new problem called Tree-Residue Vertex-Breaking (TRVB): given a multigraph G some of whose vertices are marked "breakable," is it possible to convert G into a tree via a sequence of "vertex-breaking" operations (replacing a degree-k breakable vertex by k degree-1 vertices, disconnecting the k incident edges)?

We characterize the computational complexity of TRVB with any combination of the following additional constraints: G must be planar, G must be a simple graph, the degree of every breakable vertex must belong to an allowed list B, and the degree of every unbreakable vertex must belong to an allowed list U. The two results which we expect to be most generally applicable are that (1) TRVB is polynomially solvable when breakable vertices are restricted to have degree at most 3; and (2) for any $k \geq 4$, TRVB is NP-complete when the given multigraph is restricted to be planar and to consist entirely of degree-k breakable vertices. To demonstrate the use of TRVB, we give a simple proof of the known result that Hamiltonicity in max-degree-3 square grid graphs is NP-hard.

We also demonstrate a connection between TRVB and the Hypergraph Spanning Tree problem. This connection allows us to show that the Hypergraph Spanning Tree problem in k-uniform 2-regular hypergraphs is NP-complete for any $k \geq 4$, even when the incidence graph of the hypergraph is planar.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases NP-hardness, graphs, Hamiltonicity, hypergraph spanning tree

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.32

Related Version `https://arxiv.org/abs/1706.07900`

Acknowledgements We would like to thank Zachary Abel and Jayson Lynch for their helpful discussion about this research. We would also like to thank Yahya Badran for pointing out the connection between TRVB and the Hypergraph Spanning Tree problem.

1 Introduction

In this paper, we introduce the Tree-Residue Vertex-Breaking (TRVB) problem. Given a multigraph G some of whose vertices are marked "breakable," TRVB asks whether it is possible to convert G into a tree via a sequence of applications of the *vertex-breaking*

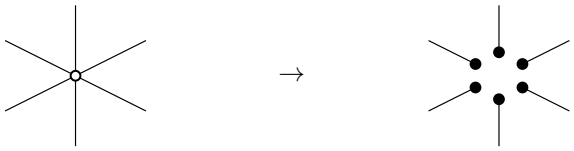


Figure 1 The operation of breaking a vertex. The vertex (left) is replaced by a set of degree-1 vertices with the same edges (right).

operation: replacing a degree-k breakable vertex with k degree-1 vertices, disconnecting the incident edges, as shown in Figure 1.

In this paper, we analyze the computational complexity of this problem as well as several variants (special cases) where G is restricted with any subset of the following additional constraints:

1. every breakable vertex of G must have degree from a list B of allowed degrees;
2. every unbreakable vertex of G must have degree from a list U of allowed degrees;
3. G is planar;
4. G is a simple graph (rather than a multigraph).

Modifying TRVB to include these constraints makes it easier to reduce from the TRVB problem to some other. For example, having a restricted list of possible breakable vertex degrees B allows a reduction to include gadgets only for simulating breakable vertices of those degrees, whereas without that constraint, the reduction would have to support simulation of breakable vertices of any degree.

We prove the following results (summarized in Table 1), which together fully classify the variants of TRVB into polynomial-time solvable and NP-complete problems:

1. Every TRVB variant whose breakable vertices are only allowed to have degrees of at most 3 is solvable in polynomial time.
2. Every planar simple graph TRVB variant whose breakable vertices are only allowed to have degrees of at least 6 and whose unbreakable vertices are only allowed to have degrees of at least 5 is solvable in polynomial time (and in fact the correct output is always "no").
3. In all other cases, the TRVB variant is NP-complete. In particular, the TRVB variant is NP-complete if the variant allows breakable vertices of some degree $k \geq 4$, and in the planar graph case, also allows either breakable vertices of some degree $b \leq 5$ or unbreakable vertices of some degree $u \leq 4$. For example, for any $k \geq 4$, TRVB is NP-complete in planar multigraphs whose vertices are all breakable and have degree k.

Among these results, we expect the most generally applicable to be the results that (1) TRVB is polynomially solvable when breakable vertices are restricted to have degree at most 3; and (2) for any $k \geq 4$, TRVB is NP-complete when the given multigraph is restricted to be planar and to consist entirely of degree-k breakable vertices.

Application to proving hardness

In general, the TRVB problem is useful when proving NP-hardness of what could be called *single-traversal problems*: problems in which some space (e.g., a configuration graph or a grid) must be traversed in a single path or cycle subject to local constraints. Hamiltonian Cycle and its variants fall under this category, but so do other problems. For example, a single traversal problem may allow the solution path/cycle to skip certain vertices entirely while mandating other local constraints. In other words, TRVB can be a useful alternative to Hamiltonian Cycle when proving NP-hardness of problems related to traversal.

Table 1 A summary of this paper's results (where B and U are the allowed breakable and unbreakable vertex degrees).

All breakable vertices have small degree ($B \subseteq \{1,2,3\}$)	**Graph restrictions**	**All vertices have large degree ($B \cap \{1,2,3,4\} = \emptyset$ and $U \cap \{1,2,3,4,5\} = \emptyset$)**	**TRVB variant complexity**	**Section**
Yes	$*$	$*$	Polynomial Time	Section 9
No	Planar or simple or unrestricted	$*$	NP-complete	Sections 4, 5, 6
No	Planar and simple	No	NP-complete	Section 7
No	Planar and simple	Yes	Polynomial Time (every instance is a "no" instance)	Section 8

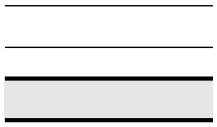


Figure 2 Abstraction of a possible edge gadget (top) and the local solution (bottom). The bold paths are (forced to be) part of the traversal while the "inside" of the gadget is shown in grey.

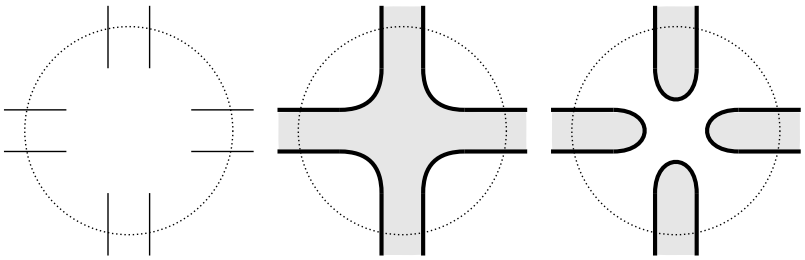


Figure 3 Abstraction of a possible breakable vertex gadget. The gadget should join some number of edge gadgets (in this case four) as shown on the left. The center and right figures show the two possible local solutions to the breakable vertex gadget. One solution connects the interiors of the incoming edge gadgets within the vertex gadget while the other disconnects them. In both figures, the bold paths are part of the traversal, while the "inside" of the gadget is shown in grey.

To prove a single-traversal problem hard by reducing from TRVB, it is sufficient to demonstrate two gadgets: an edge gadget and a breakable degree-k vertex gadget for some $k \geq 4$. This is because TRVB remains NP-hard even when the only vertices present are degree-k breakable vertices for some $k \geq 4$. Furthermore, since this version of TRVB remains NP-hard even for planar multigraphs, this approach can be used even when the single-traversal problem under consideration involves traversal of a planar space.

One possible approach for building the gadgets is as follows. The edge gadget should contain two parallel paths, both of which must be traversed because of the local constraints of the single-traversal problem (see Figure 2). The vertex gadget should have exactly two possible solutions satisfying the local constraints of the problem: one solution should disconnect the regions inside all the adjoining edge gadgets, while the other should connect these regions inside the vertex gadget (see Figure 3). We then simulate the multigraph from the input TRVB instance by placing these edge and vertex gadgets in the shape of the input multigraph as shown in Figure 4.

When trying to solve the resulting single-traversal instance, the only option (while satisfying local constraints) is to choose one of the two possible local solutions at each vertex gadget, corresponding to the choice of whether to break the vertex. The candidate solution

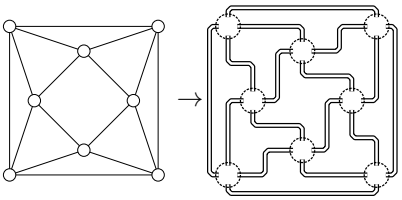


Figure 4 The input multigraph on the left could be converted into a layout of edge and vertex gadgets as shown on the right. In this example, we use a grid layout; in general, we could use any layout consistent with the edge and vertex gadgets.

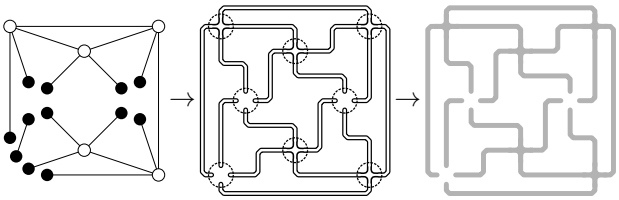


Figure 5 A choice of which vertices to break in the input multigraph (left) corresponds to a choice of local solutions at each of the breakable vertex gadgets, thereby yielding a candidate solution to the single-traversal instance (center). As a result, the shape of the interior of the candidate solution (right) is essentially the same as the shape of the residual multigraph after breaking vertices.

produced will satisfy all local constraints, but might still not satisfy the global (single cycle) constraint. Notice that the candidate solution is the boundary of the region "inside" the local solutions to the edge and vertex gadgets, and that this region ends up being the same shape as the multigraph obtained after breaking vertices. See Figure 5 for an example. The boundary of this region is a single cycle if and only if the region is connected and hole-free. Since the shape of this region is the same as the shape of the multigraph obtained after breaking vertices, this condition on the region's shape is equivalent to the condition that the residual multigraph must be connected and acyclic, or in other words, a tree. Thus, this construction yields a correct reduction, and in general this proof idea can be used to show NP-hardness of single-traversal problems.

Outline

In Section 2, we give an example of an NP-hardness proof following the above strategy. By reducing from TRVB, we give a simple proof that Hamiltonian Cycle in max-degree-3 square grid graphs is NP-hard (a result previously shown in [3]). We also use the same proof idea in manuscript [1] to show the novel result that Hamiltonian Cycle in hexagonal thin grid graphs is NP-hard.

In Section 3, we formally define the variants of TRVB under consideration. In the full version of this paper, we prove membership in NP and provide the obvious reductions between the variants.

Sections 4–7 address our NP-hardness results. In Section 4, we reduce from an NP-hard problem to show that Planar TRVB with only degree-k breakable vertices and unbreakable degree-4 vertices is NP-hard for any $k \geq 4$. All the other hardness results in this paper are derived directly or indirectly from this one. In Section 5, we prove the NP-completeness of the variants of TRVB and of Planar TRVB in which breakable vertices of some degree $k \geq 4$ are allowed. Similarly, we show in Section 6 that Graph TRVB is also NP-complete in the presence of breakable vertices of degree $k \geq 4$. Finally, in Section 7, we show that Planar Graph TRVB is NP-complete provided (1) breakable vertices of some degree $k \geq 4$ are allowed and (2) either breakable vertices of degree $b \leq 5$ or unbreakable vertices of degree $u \leq 4$ are allowed.

Next, in Section 8, we proceed to one of our polynomial-time results: that a variant of TRVB is solvable in polynomial time whenever the multigraph is restricted to be a planar graph, the breakable vertices are restricted to have degree at least 6, and the unbreakable vertices are restricted to have degree at least 5. In such a graph, it is impossible to break a set of breakable vertices and get a tree. As a result, variants of TRVB satisfying these restrictions are always solvable with a trivial polynomial time algorithm.

In Section 9, we establish a connection between TRVB and the Hypergraph Spanning Tree problem (given a hypergraph, decide whether it has a spanning tree). Namely, Hypergraph Spanning Tree on a hypergraph is equivalent to TRVB on the corresponding incidence graph with edge nodes marked breakable and vertex nodes marked unbreakable. This equivalence allows us to construct a reduction from TRVB to Hypergraph Spanning Tree: given a TRVB instance, we can first convert that instance into a bipartite TRVB instance (by inserting unbreakable vertices between adjacent breakable vertices and merging adjacent unbreakable vertices) and then construct the hypergraph whose incidence graph is the bipartite TRVB instance.

This connection allows us to obtain results about both TRVB and Hypergraph Spanning Tree. By leveraging known results about Hypergraph Spanning Tree (see [2]), we prove that TRVB is polynomial-time solvable when all breakable vertices have small degrees ($B \subseteq \{1, 2, 3\}$). This final result completes our classification of the variants of TRVB. We also apply the hardness results from this paper to obtain new results about Hypergraph Spanning Tree: namely, Hypergraph Spanning Tree is NP-complete in k-uniform 2-regular hypergraphs for any $k \geq 4$, even when the incidence graph of the hypergraph is planar. This improves the previously known result that Hypergraph Spanning Tree is NP-complete in k-uniform hypergraphs for any $k \geq 4$ (see [5]).

2 Example of how to use TRVB: Hamiltonicity in max-degree-3 square grid graphs

In this section, we show one example of using TRVB to prove hardness of a single-traversal problem. Namely, the result that Hamiltonian Cycle in max-degree-3 square grid graphs is NP-hard [3] can be reproduced with the following much simpler reduction.

The reduction is from the variant of TRVB in which the input multigraph is restricted to be planar and to have only degree-4 breakable vertices, which is shown NP-complete in Section 5. Given a planar multigraph G with only degree-4 breakable vertices, we output a max-degree-3 square grid graph by appropriately placing breakable degree-4 vertex gadgets (shown in Figure 7) and routing edge gadgets (shown in Figure 6) to connect them. The appropriate placement of gadgets can be accomplished in polynomial time by the results from [6]. Each edge gadget consists of two parallel paths of edges a distance of two apart, and as shown in the figure, these paths can turn, allowing the edge to be routed as necessary (without parity constraints). Each breakable degree-4 vertex gadget joins four edge gadgets in the configuration shown. Note that, as desired, the maximum degree of any vertex in the resulting grid graph is 3.

Consider any candidate set of edges C that could be a Hamiltonian cycle in the resulting grid graph. In order for C to be a Hamiltonian cycle, C must satisfy both the local constraint that every vertex is incident to exactly two edges in C and the global constraint that C is a cycle (rather than a set of disjoint cycles). It is easy to see that, in order to satisfy the local constraint, every edge in every edge gadget must be in C. Similarly, there are only two possibilities within each breakable degree-4 vertex gadget which satisfy the local constraint. These possibilities are shown in Figure 8.

We can identify the choice of local solution at each breakable degree-4 vertex gadget with the choice of whether to break the corresponding vertex. Under this bijection, every candidate solution C satisfying local constraints corresponds with a possible multigraph G' formed from G by breaking vertices. The key insight is that the shape of the region R inside C is exactly the shape of G'. This is shown for an example graph-piece in Figure 9.

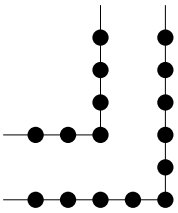


Figure 6 An edge gadget consisting of two parallel paths a distance of 2 apart.

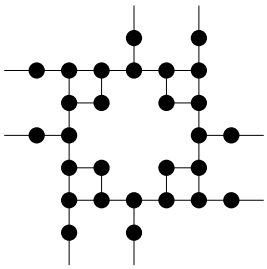


Figure 7 A degree-4 breakable vertex gadget.

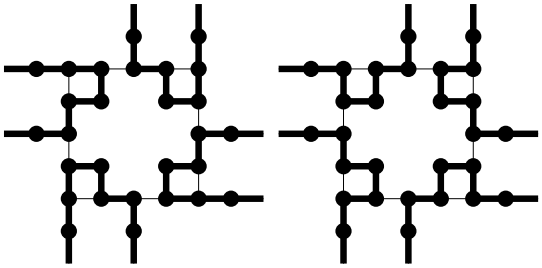


Figure 8 The two possible solutions to the vertex gadget from Figure 7 that satisfy the local constraints imposed by the Hamiltonian Cycle problem (broken on the left and unbroken on the right).

The boundary of R, also known as C, is exactly one cycle if and only if R is connected and hole-free. Since the shape of region R is the same as the shape of multigraph G', this corresponds to the condition that G' is connected and acyclic, or in other words that G' is a tree. Thus, there exists a candidate solution C to the Hamiltonian Cycle instance (satisfying the local constraints) that is an actual solution (also satisfying the global constraints) if and only if G is a "yes" instance of TRVB. Therefore, Hamiltonian Cycle in max-degree-3 square grid graphs is NP-hard.

3 Problem variants

In this section, we will formally define the variants of TRVB under consideration. In the full version of the paper, we also prove some basic results about these variants.

To begin, we formally define the TRVB problem. The multigraph operation of *breaking* vertex v in undirected multigraph G results in a new multigraph G' by removing v, adding a number of new vertices equal to the degree of v in G, and connecting these new vertices to the neighbors of v in G in a one-to-one manner (as shown in Figure 1 in Section 1). Using this definition, we pose the TRVB problem:

▶ **Problem 1.** *The* Tree-Residue Vertex-Breaking Problem (TRVB) *takes as input a multigraph G whose vertices are partitioned into two sets V_B and V_U (called the* breakable *and* unbreakable *vertices respectively), and asks to decide whether there exists a set $S \subseteq V_B$ such that after breaking every vertex of S in G, the resulting multigraph is a tree.*

In order to avoid trivial cases, we consider only input graphs that have no degree-0 vertices.

Next, suppose B and U are both sets of positive integers. Then we can constrain the breakable vertices of the input to have degrees in B and constrain the unbreakable vertices of the input to have degrees in U. The resulting constrained version of the problem is defined below:

▶ **Definition 2.** The (B, U)*-variant of the TRVB problem*, denoted (B, U)-TRVB, is the special case of TRVB where the input multigraph is restricted so that every breakable vertex in G has degree in B and every unbreakable vertex in G has degree in U.

Throughout this paper we consider only sets B and U for which membership can be computed in pseudopolynomial time (i.e., membership of n in B or U can be computed in time polynomial in n). As a result, verifying that the vertex degrees of a given multigraph are allowed can be done in polynomial time.

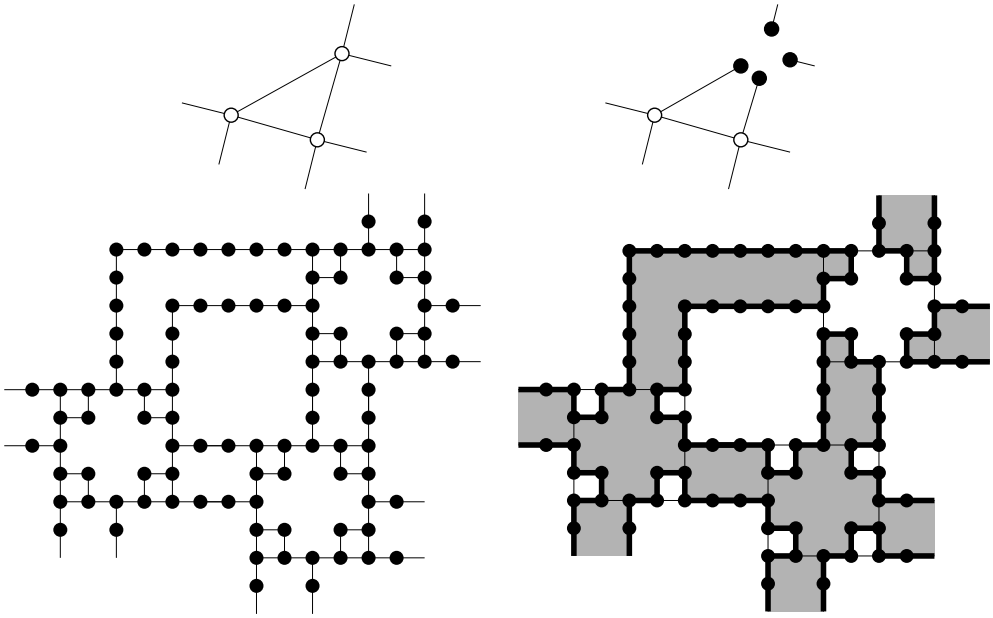


Figure 9 Given a multigraph including the piece shown in the top left, the output grid graph might include the section shown in the bottom left (depending on graph layout). If the top vertex in this piece of the multigraph is broken, resulting in the piece of multigraph G' shown in the top right, then the resulting candidate solution C (shown in bold) in the bottom right contains region R (shown in grey) whose shape resembles the shape of G'.

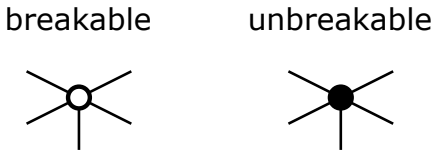


Figure 10 Depiction of vertex types in this paper.

We can also define three further variants of the problem depending on whether G is constrained to be planar, a (simple) graph, or both: the *Planar* (B,U)*-variant of the TRVB problem* (denoted Planar (B,U)-TRVB), the *Graph* (B,U)*-variant of the TRVB* (denoted Graph (B,U)-TRVB), and the *Planar Graph* (B,U)*-variant of the TRVB problem* (denoted Planar Graph (B,U)-TRVB).

3.1 Diagram conventions

Throughout this paper, when drawing diagrams, we will use filled circles to represent unbreakable vertices and unfilled circles to represent breakable vertices. See Figure 10.

4 Planar $(\{k\},\{4\})$-TRVB is NP-hard for any $k \geq 4$

The overall goal of this section is to prove NP-hardness for several variants of TRVB. In particular, we will introduce an NP-hard variant of the Hamiltonicity problem in Section 4.1 and then reduce from this problem to Planar $(\{k\},\{4\})$-TRVB for any $k \geq 4$ in Section 4.2. This is the only reduction from an external problem in this paper. All further hardness results will be derived from this one via reductions between different TRVB variants.

4.1 Planar Hamiltonicity in Directed Graphs with all in- and out-degrees 2 is NP-hard

The following problem was shown NP-complete in [4]:

▶ **Problem 3.** *The* Planar Max-Degree-3 Hamiltonicity Problem *asks for a given planar directed graph whose vertices each have total degree at most* 3 *whether the graph is Hamiltonian (has a Hamiltonian cycle).*

For the sake of simplicity we will assume that every vertex in an input instance of the Planar Max-Degree-3 Hamiltonicity problem has both in- and out-degree at least 1 (and therefore at most 2). This is because the existence of a vertex with in- or out-degree 0 in a graph immediately implies that there is no Hamiltonian cycle in that graph.

As it turns out, this problem is not quite what we need for our reduction, so below we introduce several new definitions and define a new variant of the Hamiltonicity problem:

▶ **Definition 4.** Call a vertex $v \in G$ *alternating* for a given planar embedding of a planar directed graph G if, when going around the vertex, the edges switch from inward to outward oriented more than once. Otherwise, call the vertex *non-alternating*. A non-alternating vertex has all its inward oriented edges in one contiguous section and all its outward oriented edges in another; an alternating vertex on the other hand alternates between inward and outward sections more times.

We call a planar embedding of planar directed graph G a *planar non-alternating embedding* if every vertex is non-alternating under that embedding. If G has a planar non-alternating embedding we say that G is a *planar non-alternating graph*.

▶ **Problem 5.** *The* Planar Non-Alternating Indegree-2 Outdegree-2 Hamiltonicity Problem *asks, for a given planar non-alternating directed graph whose vertices each have in- and out-degree exactly* 2*, whether the graph is Hamiltonian*

In the full version of this paper we prove that this problem is NP-hard by reducing from the Planar Max-Degree-3 Hamiltonicity Problem:

▶ **Theorem 6.** *The Planar Non-Alternating Indegree-*2 *Outdegree-*2 *Hamiltonicity Problem is NP-hard.*

4.2 Reduction to Planar $(\{k\}, \{4\})$-TRVB for any $k \geq 4$

Consider the following algorithm R_k:

▶ **Definition 7.** For $k \geq 4$, algorithm R_k takes as input a planar non-alternating graph G whose vertex in- and out-degrees all equal 2, and outputs an instance M' of Planar $(\{k\}, \{4\})$-TRVB.

To begin, we construct a labeled undirected multigraph M as follows; refer to Figure 11.

First we build all the vertices (and vertex labels) of M. For each vertex in G, we include an unbreakable vertex in M and for each edge in G we include a breakable vertex in M. If v is a vertex or e is an edge of G, we define $m(v)$ and $m(e)$ to be the corresponding vertices in M.

Next we add all the edges of M. Fix vertex v in G. Let (u_1, v) and (u_2, v) be the edges into v and let (v, w_1) and (v, w_2) be the edges out of v. Then add the following edges to M:

- Add an edge from $m(v)$ to each of $m((u_1, v))$, $m((u_2, v))$, $m((v, w_1))$, and $m((v, w_2))$.
- Add an edge from $m((v, w_1))$ to $m((v, w_2))$.
- Add $k - 3$ edges from $m((u_1, v))$ to $m((u_2, v))$.

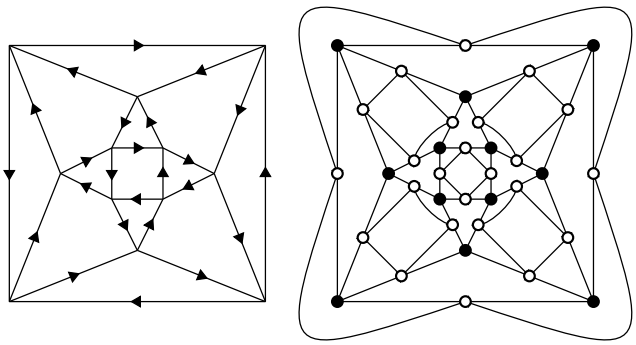


Figure 11 If the planar non-alternating directed graph on the left is G, and if $k = 4$, then we first produce multigraph M on the right. If $k > 4$, then the output M remains the same except some edges are duplicated.

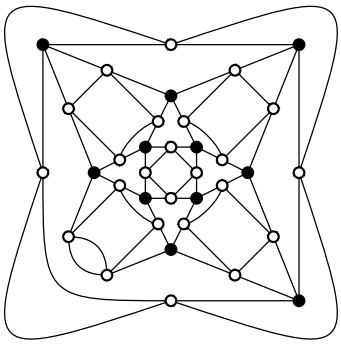


Figure 12 We modify M in the vicinity of one vertex $\hat{v}$ to get the output M' of our reduction. This figure shows one possible M' for the M in Figure 11, where $\hat{v}$ is chosen to be the bottom left vertex.

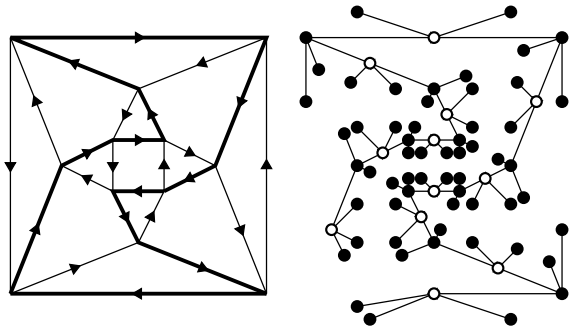


Figure 13 This figure shows a Hamiltonian cycle in example graph G from Figure 11 (left) and the corresponding solution of TRVB instance M' shown in Figure 12 (right).

Finally, pick any specific vertex $\hat{v}$ in G; refer to Figure 12. Let $(u_1, \hat{v})$ and $(u_2, \hat{v})$ be the edges into $\hat{v}$ and let $(\hat{v}, w_1)$ and $(\hat{v}, w_2)$ be the edges out of $\hat{v}$. We modify M by removing vertex $m(\hat{v})$ (and all incident edges), and adding the two edges $(m((u_1, \hat{v})), m((u_2, \hat{v})))$, and $(m((\hat{v}, w_1)), m((\hat{v}, w_2)))$. Call the resulting multigraph M' and return it as output of algorithm R_k.

We prove in the full version of this paper that algorithm R_k is a polynomial time reduction from the Planar Non-Alternating Indegree-2 Outdegree-2 Hamiltonicity Problem to Planar $(\{k\}, \{4\})$-TRVB. Figure 13 demonstrates the correspondence between a Hamiltonian Cycle in input G and a TRVB solution in output $R_k(G) = M'$. Thus we have the following:

▶ **Theorem 8.** *Planar $(\{k\}, \{4\})$-TRVB is NP-hard for any $k \geq 4$.*

5 Planar TRVB and TRVB are NP-complete with high-degree breakable vertices

▶ **Theorem 9.** *Planar (B, U)-TRVB is NP-complete if B contains any $k \geq 4$. Also (B, U)-TRVB is NP-complete if B contains any $k \geq 4$.*

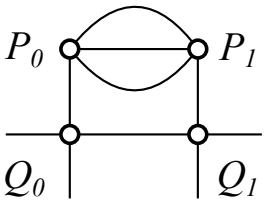


Figure 14 A gadget simulating an unbreakable degree-4 vertex using a planar arrangement of only breakable degree-4 vertices.

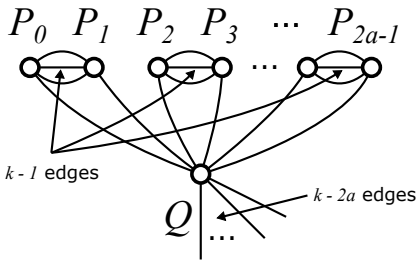


Figure 15 A gadget simulating an unbreakable degree-$(k-2a)$ vertex using only breakable degree-k vertices arranged in a planar manner. For $k > 4$, choosing a appropriately yields an unbreakable degree-3 or degree-4 gadget.

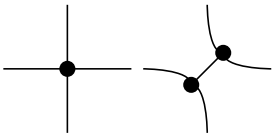


Figure 16 The degree-4 unbreakable vertex on the left can be simulated with two degree-3 unbreakable vertices as shown on the right while maintaining planarity.

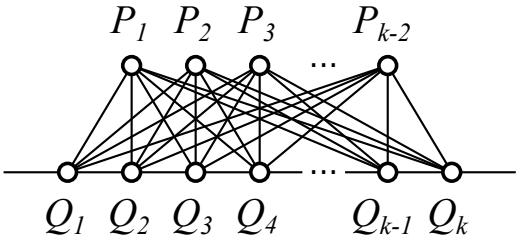


Figure 17 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-k vertices arranged without self loops or duplicated edges.

The basic idea for this theorem is to reduce from Planar $(\{k\},\{4\})$-TRVB to Planar $(\{k\},\emptyset)$-TRVB by creating a gadget which simules the behavior of an unbreakable degree-4 vertex using only breakable degree-k vertices. Figures 14, 15, and 16 sketch the construction of this gadget.

6 Graph TRVB is NP-complete with high-degree breakable vertices

▶ **Theorem 10.** *Graph (B,U)-TRVB is NP-complete if B contains any $k \geq 4$.*

The basic idea for this theorem is to reduce from (B,U)-TRVB by inserting a gadget into each edge which behaves like a degree-2 unbreakable vertices and which is built entirely out of breakable degree-k vertices. This converts the multigraph into a simple graph without affecting the answer of the TRVB instance and without adding any new values to B or U. Figure 17 sketches the construction of this gadget.

7 Planar Graph TRVB is NP-hard with both low-degree vertices and high-degree breakable vertices

▶ **Theorem 11.** *Planar Graph (B,U)-TRVB is NP-complete if (1) either $B \cap \{1,2,3,4,5\} \neq \emptyset$ or $U \cap \{1,2,3,4\} \neq \emptyset$ and (2) there exists a $k \geq 4$ with $k \in B$.*

As in the previous section, the idea for this theorem is to use unbreakable degree-2 vertex gadgets to reduce from Planar (B,U)-TRVB, converting the input multigraph into a simple graph. We build such a gadget in one of several ways, depending on which vertex types are present. Figures 18–24 sketch the gadget construction for the various cases. See the full version of this paper for details.

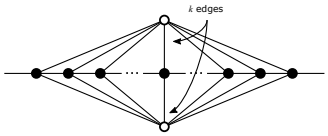


Figure 18 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-k and unbreakable degree-4 vertices arranged in a planar manner without self loops or duplicate edges.

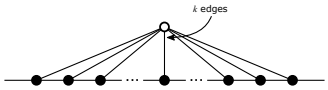


Figure 19 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-k and unbreakable degree-3 vertices arranged in a planar manner without self loops or duplicate edges.

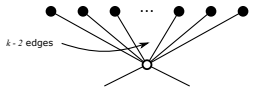


Figure 20 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-k and unbreakable degree-1 vertices arranged in a planar manner without self loops or duplicate edges.

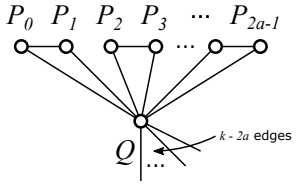


Figure 21 A gadget simulating an unbreakable degree-$(k-2a)$ vertex using only breakable degree-k and degree-2 vertices arranged in a planar manner without self loops or duplicate edges.

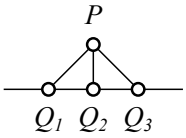


Figure 22 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-3 vertices arranged in a planar manner without self loops or duplicate edges.

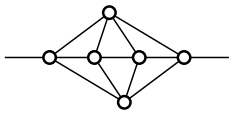


Figure 23 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-4 vertices arranged in a planar manner without self loops or duplicate edges.

8 Planar Graph TRVB is polynomial-time solvable without small vertex degrees

The overall purpose of this section is to show that variants of Planar Graph TRVB which disallow all small vertex degrees are polynomial-time solvable because the answer is always "no." Consider for example the following theorem.

▶ **Theorem 12.** *If $b > 5$ for every $b \in B$ and $u > 5$ for every $u \in U$, then Planar Graph (B,U)-TRVB has no "yes" inputs. As a result, Planar Graph (B,U)-TRVB problem is polynomial-time solvable.*

Proof. The average degree of a vertex in a planar graph must be less than 6, so there are no planar graphs with all vertices of degree at least 6. Thus, if $b > 5$ for every $b \in B$ and $u > 5$ for every $u \in U$, then every instance of Planar Graph (B,U)-TRVB is a "no" instance. ◀

In fact, we will strengthen this theorem below to disallow "yes" instances even when degree-5 unbreakable vertices are present by using the particular properties of the TRVB problem. Note that this time, planar graph inputs which satisfy the degree constraints are possible, but any such graph will still yield a "no" answer to the Tree-Residue Vertex-Breaking problem.

We describe the proof idea in Section 8.1 with details available in the full version of the paper.

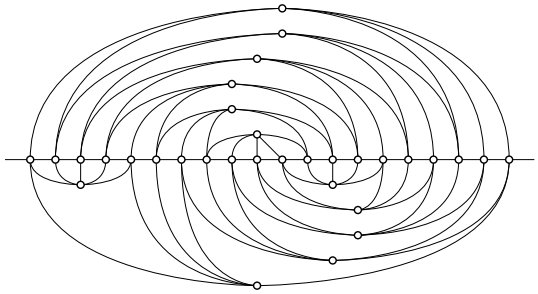


Figure 24 A gadget simulating an unbreakable degree-2 vertex using only breakable degree-5 vertices arranged in a planar manner without self loops or duplicate edges.

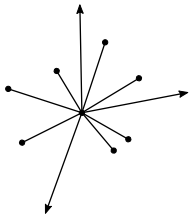


Figure 25 A degree-10 vertex with seven degree-1 neighbors (shown) and three other neighbors (not shown). The edges to the degree-1 neighbors form two bundles of size 2 and one bundle of size 3.

8.1 Proof idea

Consider the hypothetical situation in which we have a solution to the TRVB problem in a planar graph whose unbreakable vertices each have degree at least 5 and whose breakable vertices each have degree at least 6. The general idea of the proof is to show that this situation is impossible by assigning a scoring function (described below) to the possible states of the graph as vertices are broken. The score of the initial graph can easily be seen to be zero and assuming the TRVB instance has a solution, the score of the final tree can be shown to be positive. It is also the case, however, that if we break the vertices in the correct order, no vertex increases the score when broken, implying a contradiction.

Next, we introduce the scoring mechanism. Consider one vertex in the graph after some number of vertices have been broken. This vertex has several neighbors, some of which have degree 1. We can group the edges of this vertex that lead to degree-1 neighbors into "bundles" seperated by the edges leading to higher degree neighbors. For example, in Figure 25, the vertex shown has two bundles of size 2 and one bundle of size 3. Each bundle is given a score according to its size, and the score of the graph is equal to the cumulative score of all present bundles. In particular, if a bundle has a size of 1, then we assign the bundle a score of -1, and otherwise we assign the bundle a score of $n - 1$ where n is the size of the bundle.

As it turns out, under this scoring mechanism, any tree all of whose non-leaves have degree at least 5 always has a positive score. In fact, it is easy to see that in our TRVB instance, if breaking some set of breakable vertices S results in a tree, then this degree constraint applies: the non-leaves are vertices from the original graph and therefore have degree at least 5. Thus, the score of the original graph is zero (since there are no bundles), and the score after all the vertices in S are broken is positive.

Next, we define a breaking order for the vertices of S. In short, we will break the vertices of S starting on the exterior of the graph and moving inward. More formally, we

will repeatedly do the following step until all vertices in S have been broken. Consider the external face of the graph at the current stage of the breaking process. Since not every vertex in S has been broken, the graph is not yet a tree and the current external face is a cycle. Every cycle in the graph must contain a vertex from S (in order for the final graph to be a tree), so choose a vertex from S on the current external face and break that vertex next.

Breaking the vertices of S in this order has an interesting effect on the bundles in the graph: since every vertex from S is on the external face when it is broken, every degree-1 vertex ends up within the external face when it appears. Thus all bundles are within the external face of the graph at all times.

Consider the effect that breaking one vertex from S with degree $d \geq 6$ has on the score of the graph. Any vertex in S on the external face has exactly two edges which border this face. The remaining $d-2$ edges must all leave the vertex into the interior of the graph. When the vertex is broken, each of these $d-2$ edges becomes a new bundle (since the interior of the graph never has any bundles). Thus, breaking the vertex creates $d-2$ new bundles of size 1, thereby decreasing the score of the graph by $d-2$. On the other hand, the two edges which were on the external face are now each added to a bundle, thereby increasing the size of that bundle by one and increasing its score by at most two (in the case that the size was originally 1). Thus, the increase in the score of the graph due to these two edges is at most 4. In summary, breaking one vertex decreases the graph's score by $d-2 \geq 4$ and increases the graph's score by at most 4. Thus, the total score of the graph does not increase.

Since the score of the graph does not increase with any step of the process, the final result should have at most the same score as the original graph. This contradicts the fact that the tree at the end of the process has positive score while the original graph has score zero. By contradiction, we conclude that S cannot exist, giving us our desired result.

▶ **Theorem 13.** *If $b > 5$ for every $b \in B$ and $u > 4$ for every $u \in U$, then Planar Graph (B,U)-TRVB can be solved in polynomial time.*

9 TRVB and the Hypergraph Spanning Tree problem

In the full version of this paper, we demonstrate the connection between the TRVB problem and the Hypergraph Spanning Tree problem.

In particular, we reduce from (B,U)-TRVB with $B \subseteq \{1,2,3\}$ to a version of the Hypergraph Spanning Tree problem in which the hypergraphs are restricted to have only edges with at most 3 endpoints. The Hypergraph Spanning Tree problem in such hypergraphs is known to be polynomial-time solvable (see [2]), so we can conclude the following:

▶ **Theorem 14.** *(B,U)-TRVB with $B \subseteq \{1,2,3\}$ is polynomial-time solvable.*

We also reduce from Planar $(\{k\},\emptyset)$-TRVB to a version of the Hypergraph Spanning Tree problem in which the hypergraphs are restricted to be k-uniform and 2-regular and to have planar incidence graphs. Applying the fact that Planar $(\{k\},\emptyset)$-TRVB is NP-hard for any $k \geq 4$, we immediately obtain the following:

▶ **Theorem 15.** *The Hypergraph Spanning Tree problem is NP-complete in k-uniform 2-regular hypergraphs for any $k \geq 4$, even when the incidence graph of the hypergraph is planar.*

References

1 Erik D. Demaine and Mikhail Rudoy. Hamiltonicity is hard in thin or polygonal grid graphs, but easy in thin polygonal grid graphs. arXiv:1706.10046, 2017. https://arxiv.org/abs/1706.10046.

2 László Lovász. Matroid matching and some applications. *J. Comb. Theory, Ser. B*, 28(2):208–236, 1980. doi:10.1016/0095-8956(80)90066-0.

3 Christos H. Papadimitriou and Umesh V. Vazirani. On two geometric problems related to the traveling salesman problem. *J. Algorithms*, 5(2):231–246, 1984. doi:10.1016/0196-6774(84)90029-4.

4 Ján Plesník. The np-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201, 1979. doi:10.1016/0020-0190(79)90023-1.

5 Hans Jürgen Prömel and Angelika Steger. *The Steiner tree problem: a tour through graphs, algorithms, and complexity*. Springer Science & Business Media, 2002.

6 Markus W. Schäffter. Drawing graphs on rectangular grids. *Discrete Applied Mathematics*, 63(1):75–89, 1995. doi:10.1016/0166-218X(94)00020-E.

Nearly Optimal Separation Between Partially and Fully Retroactive Data Structures

Lijie Chen[1]
Massachusetts Institute of Technology
lijieche@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology
edemaine@mit.edu

Yuzhou Gu
Massachusetts Institute of Technology
yuzhougu@mit.edu

Virginia Vassilevska Williams[2]
Massachusetts Institute of Technology
virgi@mit.edu

Yinzhan Xu
Massachusetts Institute of Technology
xyzhan@mit.edu

Yuancheng Yu
Massachusetts Institute of Technology
ycyu@mit.edu

Abstract

Since the introduction of retroactive data structures at SODA 2004, a major unsolved problem has been to bound the gap between the best partially retroactive data structure (where changes can be made to the past, but only the present can be queried) and the best fully retroactive data structure (where the past can also be queried) for any problem. It was proved in 2004 that any partially retroactive data structure with operation time $T_{\mathsf{op}}(n, m)$ can be transformed into a fully retroactive data structure with operation time $O(\sqrt{m} \cdot T_{\mathsf{op}}(n, m))$, where n is the size of the data structure and m is the number of operations in the timeline [7]. But it has been open for 14 years whether such a gap is necessary.

In this paper, we prove nearly matching upper and lower bounds on this gap for all n and m. We improve the upper bound for $n \ll \sqrt{m}$ by showing a new transformation with multiplicative overhead $n \log m$. We then prove a lower bound of $\min\{n \log m, \sqrt{m}\}^{1-o(1)}$ assuming any of the following conjectures:

- **Conjecture I:** Circuit SAT requires $2^{n-o(n)}$ time on n-input circuits of size $2^{o(n)}$.

  This conjecture is far weaker than the well-believed SETH conjecture from complexity theory, which asserts that CNF SAT with n variables and $O(n)$ clauses already requires $2^{n-o(n)}$ time.

- **Conjecture II:** Online $(\min, +)$ product between an integer $n \times n$ matrix and n vectors requires $n^{3-o(1)}$ time.

  This conjecture is weaker than the APSP conjectures widely used in fine-grained complexity.

[1] Supported by an Akamai Fellowship.
[2] Partially supported by an NSF Career Award, a Sloan Fellowship, NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018).
Editor: David Eppstein; Article No. 33; pp. 33:1–33:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- **Conjecture III (3-SUM Conjecture):** Given three sets A, B, C of integers, each of size n, deciding whether there exist $a \in A, b \in B, c \in C$ such that $a + b + c = 0$ requires $n^{2-o(1)}$ time.

  This 1995 conjecture [13] was the first conjecture in fine-grained complexity.

Our lower bound construction illustrates an interesting power of fully retroactive queries: they can be used to quickly solve batched pair evaluation. We believe this technique can prove useful for other data structure lower bounds, especially dynamic ones.

2012 ACM Subject Classification Theory of computation → Lower bounds and information complexity

Keywords and phrases retroactive data structure, conditional lower bound

Digital Object Identifier 10.4230/LIPIcs.SWAT.2018.33

Acknowledgements We would like to thank Quanquan Liu and Ryan Williams for helpful discussions, and the anonymous reviewers for their generous comments.

1 Introduction

Retroactive Data Structures

A data structure can be thought of as a sequence of updates being applied to an initial state. In traditional data structures, we can only append updates to the end of this sequence, called the *timeline*, and can only query about the final state of the data structure resulting from all the updates. *Retroactive data structures*, introduced at SODA 2004 [7], allow us to add or remove updates in the past, i.e., anywhere in the timeline rather than only at the end.

There are two main kinds of retroactive data structures: *partially retroactive* data structures, where we are only allowed to query the present, i.e., the final version resulting from the whole update sequence; and *fully retroactive* data structures, where we are also allowed to query about a past state, i.e., the state resulting from applying only a *prefix* of the update sequence given by the timeline.

Unlike persistence [11], there is no general efficient transformation from a data structure into a retroactive data structure, even partially retroactive with sublinear multiplicative overhead [7]. Nonetheless, several efficient retroactive data structures have been developed [8, 5, 15, 10, 23, 17, 24, 9].

Motivation: Full Retroactivity versus Partial Retroactivity

A key problem, posed in the original paper on retroactive data structures [7], is whether the full retroactivity requirement makes problems much harder than their partially retroactive counterpart. The same paper established an $O(\sqrt{m})$ multiplicative overhead transformation from a partially retroactive data structure to a fully retroactive one, where m is the number of updates in the timeline.

Prior to our work, there was no data structure problem whose best known fully retroactive version was substantially (more than a polylogarithmic factor) worse than the best known partially retroactive version. Priority queues *used to be* the only problem with a polynomial gap (between $O(\sqrt{m} \log m)$ and $O(\log m)$ time [7]). But at WADS 2015 it was shown that priority queues have a polylogarithmic fully retroactive solution [9], and more generally, any "time-fusable" data structure can be transformed from partial to full retroactivity with polylogarithmic overhead. *Can this transformation be generalized to all data structures?*

Our Results: Conditional Lower Bounds

We show that, perhaps surprisingly, the $O(\sqrt{m})$ overhead for transforming partial retroactivity into full retroactivity is nearly optimal for general data structure problems, conditioned on any of three well-believed conjectures:

▶ **Conjecture I.** *In the Word-RAM model of computation with $O(\log n)$ bit words, it takes $2^{n-o(n)}$ time to solve SIZE$(2^{o(n)})$ Circuit SAT: decide whether a given n-input circuit C of size $2^{o(n)}$ is satisfiable.*

▶ **Remark 1.** *The problem SIZE$(2^{o(n)})$ Circuit SAT is far harder than CNF SAT, and the conjecture above is much weaker than the well-believed Strong Exponential Time Hypothesis (SETH) [21] which states that for every $\varepsilon > 0$, there is a clause length k such that k-SAT on n variables cannot be solved in $2^{(1-\varepsilon)n}$ time. Due to the Sparsification Lemma [21], the formulas that SETH concerns have* linear *size. It is much easier to believe that Circuit SAT for an unrestricted circuit (as opposed to a formula), of much larger, $2^{o(n)}$ size requires enumeration of all possible inputs.*

▶ **Conjecture II.** *Online $(\min, +)$ product between an integer $n \times n$ matrix and n length-n vectors requires $n^{3-o(1)}$ time in the word-RAM model of computation with $O(\log n)$ bit words. That is, given an integer matrix $A \in \mathbb{Z}^{n\times n}$, and n vectors $v^1, v^2, \ldots, v^n$ that are revealed one by one, we wish to compute the $(\min, +)$-products*

$$A \diamond v \quad := \quad \left(\min_{k=1}^{n}(A_{1,k} + v_k), \quad \min_{k=1}^{n}(A_{2,k} + v_k), \quad \ldots, \quad \min_{k=1}^{n}(A_{n,k} + v_k) \right)$$

between A and each of the v^is. We get to access v^{i+1} only after we have output $A \diamond v^i$. The conjecture asserts that the whole computation requires $n^{3-o(1)}$ time.

▶ **Remark 2.** *The* offline *(and thus easier) version of the above problem is equivalent to calculating the $(\min, +)$-product of two matrices of size $n \times n$, which is known to be asymptotically equivalent to the famous* APSP *problem [12]: $(\min, +)$-product is in $O(n^c)$ time if and only if* APSP *is in $O(n^c)$ time, for any constant c.*

The online $(\min, +)$-product conjecture is a natural generalization of the online Boolean Matrix-Vector Product conjecture of Henzinger et al. [19] that asserts that given a Boolean $n \times n$ matrix, multiplying it with n Boolean vectors given online requires $n^{3-o(1)}$ time, in the Word-RAM model. There is no known relationship between the APSP conjecture and the Online Boolean Matrix-Vector Product conjecture, so one may be true even if the other fails. It is not hard to embed Boolean product into $(\min, +)$-product, and hence our conjecture is a weakening of both of these conjectures simultaneously, making ours very believable.

▶ **Conjecture III** (3-SUM Conjecture)**.** *There exists a constant q, so that given three size-n sets A, B, C of integers in $[-n^q, n^q]$, deciding whether there exist $a \in A$, $b \in B$, $c \in C$ such that $a + b + c = 0$ requires $n^{2-o(1)}$ time in the word-RAM model with $O(\log n)$ bit words.*

▶ **Remark 3.** *The 3-SUM Conjecture was the first attempt to address fine-grained complexity, back in 1995 [13]. By a standard hashing trick, we can assume $q \leq 3 + \delta$ for any $\delta > 0.3$ [26]. It remains open despite several slightly subquadratic algorithms [4, 6, 18].*

We can now state our lower bounds conditioned on the conjectures above, whose proofs are in Section 2. As in our conjectures above, throughout the paper, we assume that we are working in the word-RAM model with word size $w = \Theta(\log \max\{n, m\})$, where n denotes the size of the data structure problem and m denotes the length of the update sequence (timeline).

▶ **Theorem 1.** *There is a data structure problem that has an $O(n^{1+o(1)})$-time partially retroactive data structure, but conditioned on Conjecture I, requires $\Omega(n^{2-o(1)})$ time for fully retroactive queries when $m = \Theta(n^2)$.*

▶ **Theorem 2.** *There is a data structure problem that has an $O(\log n)$-time partially retroactive data structure, but conditioned on Conjecture II, requires $\Omega(n^{1-o(1)})$ time for fully retroactive queries when $m = \Theta(n^2)$.*

▶ **Theorem 3.** *There is a data structure problem that has an $O(\sqrt{n})$-time partially retroactive data structure, but conditioned on Conjecture III, requires $\Omega(n^{1-o(1)})$ time for fully retroactive queries when $m = \Theta(n)$.*

Our Results: Matching Upper bound

The three theorems above show that improving the general dependence on $\sqrt{m}$ is impossible based on any of these three conjectures. But we may hope to have a better data structure when $m \gg n^2$. In fact, we show in Section 3 that this is possible, for any "reasonable" data structure, by establishing the following theorem:

▶ **Theorem 4.** *Suppose a data structure of size n satisfies the following conditions:*

1. *There is a sequence of $O(n)$ queries to extract the whole state[3] $\mathcal{S}$ from it.*
2. *Given a state $\mathcal{S}$ of size n, there is a sequence of $O(n)$ operations to update the data structure from empty initial state to $\mathcal{S}$.*
3. *It is partially retroactive with operation time $T_{\mathsf{op}}(n, m)$.*

Then the corresponding problem has an amortized fully retroactive data structure with operation time $O\left(\min\{\sqrt{m}, n \log m\} \cdot T_{\mathsf{op}}(n, m)\right)$.

▶ **Remark 4.** *The data structure of Theorem 4 is similar to the data structure described in [9, Section 2.2].*

Combining the above four theorems, we conclude that under reasonable conditions, the optimal gap between partial and full retroactivity is $\Theta(\min\{\sqrt{m}, n\})$, up to $m^{o(1)}$ factors, for any n and m.

Related Work

The field of fine-grained complexity studies the exact running time for problems in P and beyond, and proves many lower bounds for data structure problems conditioned on various conjectures [25, 3, 19, 22, 1, 20, 16]. Look at the recent survey [26] for a summary of the known results in fine-grained complexity. We mention two of the related papers. Building on work by Patrascu [25] who focused on the 3-SUM conjecture, Abboud and Vassilevska W. [3] proved hardness for data structure problems under a variety of hypotheses: SETH, 3-SUM, APSP etc. [3] introduced SETH as a hardness hypothesis for data structure problems and obtained SETH-hardness for the following dynamic problems: maintaining under edge updates (insertions or deletions) the strongly connected components of a graph, the number of nodes reachable from a fixed source, a 1.3-approximation of the diameter of the graph, or whether there is $(s, t) \in S \times T$ such that s can reach t for two fixed node sets S and T. Henzinger et al. [19] introduces the Online Matrix-Vector Multiplication Conjecture, and shows that it implies tight hardness result for subgraph connectivity, Pagh's problem, d-failure connectivity, decremental single-source shortest paths, and decremental transitive closure.

[3] The state of a data structure is a description of all data it currently stores.

2 Lower Bounds

In this section, we first give a data structure framework, which eases the construction of our separation, and then we prove Theorem 1, Theorem 2, and Theorem 3.

2.1 Data Structure Framework

We present a data structure framework which turns out to be easy for partially retroactive data structures, but hard for their fully retroactive counterparts. In this framework, a data structure $\mathcal{D}$ maintains several lists, and answers a certain question on them. The formal definition is given below.

▶ **Framework 1** (Data Structure Problem $\mathcal{P}_F$)**.** *In our data structure problem $\mathcal{P}_F$. We are required to maintain a constant number of lists consisting of items from an entry set $\mathcal{E}$. Denote the lists as $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k$, and F is a function defined on these lists.*

We can view each list $\mathcal{L}_i$ as a mapping from $\mathbb{N}$ to $\mathcal{E}$ and initially every list maps all indices to the idle symbol $\perp$. We use $\mathcal{L}[a]$ to denote the a-th element of the list $\mathcal{L}$, and we measure the size of a list $\mathcal{L}$ (denoted by $|\mathcal{L}|$) by the number of a's such that $\mathcal{L}[a] \neq \perp$. The size of the data structure is then measured by sum of the sizes of all its lists.[4]

There are two types of operations.

- *set-element$(\mathcal{L}_i, a, e)$: Set $\mathcal{L}_i[a] = e$.*
- *F-evaluation: Evaluate F on the current maintained lists $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k$.*

The key property for the problem $\mathcal{P}_F$ is that, once we have a data structure $\mathcal{D}_F$ for it, it supports partially retroactive queries with essentially no overhead.

▶ **Lemma 1.** *Suppose there is a data structure $\mathcal{D}_F$ for the data structure problem $\mathcal{P}_F$ with update time T_U and query time T_Q. Then there is a partially retroactive data structure $\mathcal{D}_F^{\mathsf{part}}$ for problem $\mathcal{P}_F$ with update time $T_U + O(\log m)$ and query time T_Q.*

Proof. Our partially retroactive data structure $\mathcal{D}_F^{\mathsf{part}}$ simply simulates an instance of the regular structure $\mathcal{D}_F$ which represents the current version of the data structure. Whenever there is an update in the history, it could be either inserting or deleting an operation *set-element*$(\mathcal{L}_i, a, e)$ at time t, it only affects the a-th element in $\mathcal{L}_i$ of the current version of the data structure $\mathcal{D}_F$.

Therefore, we can use a BST to organize all *set-element* operations on each location of each list in the chronological order. We update the corresponding BST on the a-th element of list $\mathcal{L}_i$ when dealing with insertion or deletion of an operation *set-element*$(\mathcal{L}_i, a, e)$ in the history. When the latest *set-element* changes in the BST (or the BST becomes empty), we update the corresponding value in $\mathcal{D}_F$. And the query operation is equivalent to the same query operation on the current data structure $\mathcal{D}_F$. The time cost is the usual time cost of BST. ◀

2.2 Lower Bound from SIZE$(2^{o(n)})$ Circuit SAT

Now we are ready to prove our lower bounds. First we prove Theorem 1, which we repeat here for completeness:

[4] A list can also be viewed as a dictionary over integers. We view them as lists because, in our construction, it is much more convenient to do so.

▶ **Reminder: Conjecture I.** *In the Word-RAM model with $O(\log n)$ bit words, it takes $2^{n-o(n)}$ time to solve Circuit SAT on n-input circuits of size $2^{o(n)}$.*

▶ **Reminder: Theorem 1.** *There is a data structure problem that has an $O(n^{1+o(1)})$-time partially retroactive data structure, but conditioned on Conjecture I, requires $\Omega(n^{2-o(1)})$ time for fully retroactive queries when $m = \Theta(n^2)$.*

Proof. Let $d = n^{o(1)}$. We use the entry set

$$\mathcal{E} := \mathcal{C}_d \times \{0,1\}^{\leq d},$$

where $\mathcal{C}_d$ is the set of descriptions of all circuits of size at most d, and $\{0,1\}^{\leq d}$ is the set of binary strings of length at most d. These descriptions take at most $O(\text{poly}(d)) = n^{o(1)}$ bits. Therefore, an item from $\mathcal{E}$ consists of $n^{o(1)}$ bits. Denote this number by d'.

Consider the data structure problem $\mathcal{P}_{F^{(\mathsf{SAT})}}$ with respect to two lists $\mathcal{L}_1, \mathcal{L}_2$ of items in $\mathcal{E}$ and the function $F^{(\mathsf{SAT})}$ defined on them as follows. $F^{(\mathsf{SAT})}(\mathcal{L}_1, \mathcal{L}_2) = 1$ if the following holds:

> There exist a and b with $\mathcal{L}_1[a] = (C_1, x_1) \neq \perp$ and $\mathcal{L}_2[b] = (C_2, x_2) \neq \perp$ such that
> - $C_1 = C_2$;
> - C_2 is a valid description of a circuit of size at most d with exactly $|x_1| + |x_2|$ bits of input;
> - $C_2(x_1, x_2) = 1$.

$F^{(\mathsf{SAT})}(\mathcal{L}_1, \mathcal{L}_2) = 0$ otherwise. We say a pair (C_1, x_1) and (C_2, x_2) is *good* if they satisfy the conditions above.

Let $\ell := (|\mathcal{L}_1| + |\mathcal{L}_2|)$. The size of the whole structure is $n = d'\ell$.

Partially Retroactive Upper Bound. In order to maintain $F^{(\mathsf{SAT})}(\mathcal{L}_1, \mathcal{L}_2)$, we keep a counter n_{SAT} recording the number of pairs a and b such that $\mathcal{L}_1[a]$ and $\mathcal{L}_2[b]$ is a good pair. Whenever we modify an element in lists $\mathcal{L}_1$ or $\mathcal{L}_2$, it takes $O(n^{1+o(1)})$ time to update the counter n_{SAT}.

Now, since we have an $O(n^{1+o(1)})$ update time algorithm for $\mathcal{P}_{F^{(\mathsf{SAT})}}$, by Lemma 1, it extends to an algorithm for the partially retroactive version.

Fully Retroactive Lower Bound. Given a circuit C of size $2^{o(u)}$ with u inputs. Let $\ell = 2^{u/4}$ be the size of the lists in the data structure (assuming u is divisible by 4 for simplicity).

Let A and B be two identical lists of entries in $\mathcal{E}$ with size $2^{u/2} = \ell^2$, such that the i-th element of A and B is (C, w_i), where w_i is the i-th length $u/2$ binary string in lexicographic order. Then we divide A and B into $\ell = 2^{u/4}$ groups of equal size, and denote them by $A_1, A_2, \ldots, A_\ell$ and $B_1, B_2, \ldots, B_\ell$ correspondingly, where each A_i and each B_i is a list of size ℓ.

The circuit C is satisfiable if and only if there exists $a \in A$ and $b \in B$ such that a and b is a good pair. Consider the following operation sequences:

- First, for each $k \in [\ell]$, we add an operation *set-element*$(\mathcal{L}_1, k, \perp)$. We denote the operation time by t_k.
- Next for each $j \in [\ell]$, we add an operation *set-element*$(\mathcal{L}_2, k, B_j[k])$ for each $k \in [\ell]$. We denote the time right after adding the last operation for each j (*set-element*$(\mathcal{L}_2, \ell, B_j[\ell])$) by q_j.

- Now, for each $i \in [\ell]$, we replace the operation on time t_k by an operation *set-element*($\mathcal{L}_1$, $k, A_i[k]$) for each $k \in [\ell]$, and after that, we make fully retroactive query $F^{(\mathsf{SAT})}$-*evaluation* at time q_j for each $j \in [\ell]$. From the definition of $F^{(\mathsf{SAT})}$, it tells us whether there exists $a \in A_i$, $b \in B_j$ such that a and b is a good pair, for each $i, j \in [\ell]$.

The whole procedure consists of $m = \Theta(\ell^2) = O(n^2)$ operations. Conditioning on Conjecture I, the whole sequence takes at least $2^{u(1-o(1))} = \ell^{4-o(1)} = n^{4-o(1)}$ time, which means a fully retroactive operation takes at least amortized $\Omega(n^{2-o(1)})$ time, and completes the proof. ◀

2.3 Lower Bounds from Online $(\min, +)$-product

Next we prove Theorem 2, which we recap here for completeness:

▶ **Reminder: Conjecture II.** *Online* $(\min, +)$ *product between an integer* $n \times n$ *matrix and* n *length-*n *vectors requires* $n^{3-o(1)}$ *time in the word-RAM model with* $O(\log n)$ *bit words. That is, given an integer matrix* $A \in \mathbb{Z}^{n \times n}$*, and* n *vectors* $v^1, v^2, \ldots, v^n$ *which are revealed one by one, we wish to compute the* $(\min, +)$*-product*

$$A \diamond v \quad := \quad \left(\min_{k=1}^{n}(A_{1,k} + v_k), \quad \min_{k=1}^{n}(A_{2,k} + v_k), \quad \ldots, \quad \min_{k=1}^{n}(A_{n,k} + v_k) \right)$$

between A *and each of the* v^i*s. We get to access* v^{i+1} *only after we have output* $A \diamond v^i$*. The conjecture asserts that the whole computation requires* $n^{3-o(1)}$ *time.*

▶ **Reminder: Theorem 2.** *There is a data structure problem that has an* $O(\log n)$*-time partially retroactive data structure, but conditioned on Conjecture II, requires* $\Omega(n^{1-o(1)})$ *time for fully retroactive queries when* $m = \Theta(n^2)$*.*

Proof. Let c be a constant such that all entries from A and all v^i's lie in $[0, n^c]$.

Now, consider the data structure problem $\mathcal{P}_{F^{(\min,+)}}$ with respect to two lists $\mathcal{L}_1, \mathcal{L}_2$ and the function $F^{(\min,+)}$ defined on them as

$$F^{(\min,+)}(\mathcal{L}_1, \mathcal{L}_2) := \min_{a:\mathcal{L}_1[a]\neq\perp, \mathcal{L}_2[a]\neq\perp} (\mathcal{L}_1[a] + \mathcal{L}_2[a]) .$$

The entry set $\mathcal{E}$ here is the integers in $[0, n^c]$.

Partially Retroactive Upper Bound. Clearly, the operations in $\mathcal{P}_{F^{(\min,+)}}$ can be supported in $O(\text{polylog}(n))$ time: we use a priority queue to maintain the sums $\mathcal{L}_1[a] + \mathcal{L}_2[a]$ for all the valid a's, and update the priority queue correspondingly after each *set-element* operations. Therefore, by Lemma 1, we know the update/query operations in the partially retroactive version of $\mathcal{P}_{F^{(\min,+)}}$ can be supported in $O(\text{polylog}(n) + \log m)$ time.

Fully Retroactive Lower Bound. Let $a_1, a_2, \ldots, a_n$ be the n rows of A, and v be a vector. Computing the $(\min, +)$ product of A and v is equivalent to compute

$$(a_i \diamond v) := \min_{k=1}^{n} (a_{i,k} + v_k)$$

for each $i \in [n]$.

We are going to show that a fully retroactive algorithm for $\mathcal{P}_{F^{(\min,+)}}$ can be utilized to compute $(a_i \diamond v^j)$ for each $i, j \in [n]$ in an online fashion.

Consider the following operation sequences. First we add *set-element*($\mathcal{L}_1, k, 0$) for each $k \in [n]$; then for each $j \in [n]$, we add *set-element*($\mathcal{L}_2, k, a_{j,k}$) for each $k \in [n]$. We use t_j to

denote the time right after adding the operation *set-element*$(\mathcal{L}_2, n, a_{j,n})$, i.e., the time we have just set $\mathcal{L}_2$ to represent vector a_j.

Then for each $i \in [n]$, we delete the first n operations in the history (that is, we clear all the *set-element* operations on $\mathcal{L}_1$); and then we add *set-element*$(\mathcal{L}_1, k, v^i_k)$ for each $k \in [n]$ at the beginning of the operation sequence (that is, we set $\mathcal{L}_1$ to represent the vector v^i); next we make a fully retroactive query $F^{(\min,+)}$*-evaluation* at the time t_j for each $j \in [n]$. It is easy to see that querying at time t_j gives us the value of $(a_j \diamond v^i)$. So, after performing the above procedure for v^i, we have calculated the $(\min, +)$ product between A and v^i.

The size of data structure is $\Theta(n)$, and there are $m = \Theta(n^2)$ operations in total. Hence, conditioned on Conjecture II, any fully retroactive data structure running on the above algorithm takes at least amortized $n^{1-o(1)}$ time for either update or query operation. ◀

2.4 Lower Bounds from 3-SUM

Next, we prove Theorem 3, which we recap here for completeness:

▶ **Reminder: Conjecture III (3-SUM Conjecture).** *There is a constant q such that, given three size-n sets A, B, C of integers in $[-n^q, n^q]$, deciding whether there exist $a \in A$, $b \in B$, $c \in C$ such that $a + b + c = 0$ requires $n^{2-o(1)}$ time in the word-RAM model with $O(\log n)$ bit words.*

▶ **Reminder: Theorem 3.** *There is a data structure problem that has an $O(\sqrt{n})$-time partially retroactive data structure, but conditioned on Conjecture III, requires $\Omega(n^{1-o(1)})$ time for fully retroactive queries when $m = \Theta(n)$.*

Proof. Consider the data structure problem $\mathcal{P}_{F^{(3\text{SUM})}}$ with respect to three lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ and the function $F^{(3\text{SUM})}$ defined on them as follows

$$F^{(3\text{SUM})}(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3) := \begin{cases} 1 & |\mathcal{L}_2|^2 \le |\mathcal{L}_1|,\ |\mathcal{L}_3|^2 \le |\mathcal{L}_1|, \text{ and there exist } a, b, c \text{ such that} \\ & \mathcal{L}_1[a] \neq \perp,\ \mathcal{L}_2[b] \neq \perp,\ \mathcal{L}_3[c] \neq \perp \text{ and } \mathcal{L}_1[a] + \mathcal{L}_2[b] + \mathcal{L}_3[c] = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Let $n := \sum_{i=1}^{3} |\mathcal{L}_i|$ be size of the whole structure, and $n_i := |\mathcal{L}_i|$.

Partially Retroactive Upper Bound. We use $\widetilde{\mathcal{L}}_2$ (resp. $\widetilde{\mathcal{L}}_3$) to denote the sublists consisting of the first (at most) $\sqrt{n_1}$ elements of $\mathcal{L}_2$ (resp. $\mathcal{L}_3$). Then by maintaining a BST for each list, an operation on $\mathcal{L}_2$ (resp. $\mathcal{L}_3$) can be easily reduced to at most one operation on $\widetilde{\mathcal{L}}_2$ (resp. $\widetilde{\mathcal{L}}_3$). Since whenever $\widetilde{\mathcal{L}}_2 \neq \mathcal{L}_2$ or $\widetilde{\mathcal{L}}_3 \neq \mathcal{L}_3$, $F(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$ is defined to be zero, we can pretend to work with $\widetilde{\mathcal{L}}_2$ and $\widetilde{\mathcal{L}}_3$.

We build a hash table $\mathcal{H}$ storing all the elements in $\mathcal{L}_1$, and every value of the form $-a - b$ for $a \in \widetilde{\mathcal{L}}_2$, $b \in \widetilde{\mathcal{L}}_3$. Using this table, we can count and maintain the number of the triples (a, b, c) such that $\mathcal{L}_1[a] + \widetilde{\mathcal{L}}_2[b] + \widetilde{\mathcal{L}}_3[c] = 0$. We denote this number by n_{triple}.

Whenever we modify the list $\mathcal{L}_1$, we make the corresponding change on $\mathcal{H}$. This may also cause $O(1)$ additional operations on $\widetilde{\mathcal{L}}_2$ and $\widetilde{\mathcal{L}}_3$, as n_1 can be larger or smaller. And when we modify the list $\widetilde{\mathcal{L}}_2$ or $\widetilde{\mathcal{L}}_3$, this causes updating at most $\max(|\widetilde{\mathcal{L}}_2|, |\widetilde{\mathcal{L}}_3| = O(\sqrt{n})$ values in $\mathcal{H}$.

Since we have an $O(\sqrt{n})$ update time algorithm for $\mathcal{P}_{F^{(3\text{SUM})}}$, by Lemma 1, it extends to an algorithm for the partially retroactive version.

Fully Retroactive Lower Bound. Let A, B, C be three integer lists of size n. For convenience we assume that n is a square number. We divide B and C into $\sqrt{n}$ groups of

equal size, and denote them by $B_1, B_2, \ldots, B_{\sqrt{n}}$ and $C_1, C_2, \ldots, C_{\sqrt{n}}$ correspondingly. Then each B_i and each C_i is a list of size $\sqrt{n}$.

Consider the following operation sequence.

- First, for each $i \in [n]$, we add an operation *set-element*$(\mathcal{L}_1, i, A[i])$, that is, we set the list $\mathcal{L}_1$ to represent the set A; then for each $k \in [\sqrt{n}]$, we add an operation *set-element*$(\mathcal{L}_2, k, 0)$, whose operation time is denoted by t_k.
- Next for each $j \in [\sqrt{n}]$, we add an operation *set-element*$(\mathcal{L}_3, k, C_j[k])$ for each $k \in [\sqrt{n}]$. We denote the time right after adding the operation *set-element*$(\mathcal{L}_3, \sqrt{n}, C_j[\sqrt{n}])$ as time q_j.
- Now, for each $i \in [\sqrt{n}]$, we replace the operation on time t_k by an operation *set-element*$(\mathcal{L}_2, k, B_i[k])$. After that, we make a fully retroactive query F^{3SUM}-*evaluation* at time q_j for each $j \in [\sqrt{n}]$. From the definition of F^{3SUM}, the queries tell us whether there exists $a \in A$, $b \in B_i$, $c \in C_j$ such that $a + b + c = 0$ for each $i, j \in [\sqrt{n}]$, and thus solve the 3SUM problem.

The data structure above has size $\Theta(n)$, and the whole procedure consists of $m = \Theta(n)$ operations. Therefore, conditioned on Conjecture III, either update or query for a fully retroactive data structure for problem $\mathcal{P}_{F^{(3\text{SUM})}}$ takes amortized $\Omega(n^{1-o(1)})$ time. ◀

3 Upper Bounds

In this section, we prove Theorem 4:

▶ **Reminder: Theorem 4.** *Suppose a data structure of size n satisfies the following conditions:*

1. *There is a sequence of $O(n)$ queries to extract the whole state $\mathcal{S}$ from it.*
2. *Given a state $\mathcal{S}$ of size n, there is a sequence of $O(n)$ operations to update the data structure from empty initial state to $\mathcal{S}$.*
3. *It is partially retroactive with operation time $T_{\mathsf{op}}(n, m)$.*

Then the corresponding problem has an amortized fully retroactive data structure with operation time $O\left(\min\{\sqrt{m}, n \log m\} \cdot T_{\mathsf{op}}(n, m)\right)$.

Proof. We use a weight-balanced binary tree (WBT) $\mathcal{T}$ to maintain the whole operation sequence [14]. The subtree of each node u corresponds to an interval of operations S_u in the whole operation sequence. We can build a partially retroactive data structure $\mathcal{D}_u$ on S_u as augmented information in node u. One property of WBT is that when we insert or delete its nodes, the amortized total number of element changes to all S_u is only $O(\log m)$. More formally, if S_u is the set of operations before a node insertion or deletion, and S'_u is the set of operations after the insertion or deletion, then WBT ensures

$$\sum_u |S_u \setminus S'_u| + |S'_u \setminus S_u|$$

is amortized $O(\log m)$. For each element change in S_u, we can update $\mathcal{D}_u$ using the partially retroactive data structure in $O(T_{\mathsf{op}}(n, m) \cdot \log m)$ amortized time per insert/delete of an operation.

For each fully retroactive query, we first extract the corresponding prefix of the operation sequence from the WBT. By properties of WBT, in $O(\log m)$ time, we can get $k = O(\log m)$ nodes, $u_1, u_2, \ldots, u_k$, such that the concatenation of these S_{u_i}'s is exactly the prefix we are asking. Next we maintain a data structure state $\mathcal{S}$ initialized as the empty state. We go through each u_i in order: first append $O(n)$ operations at the beginning of $\mathcal{D}_u$ to set the initial state inside $\mathcal{D}_u$ to be $\mathcal{S}$, and then make $O(n)$ queries on $\mathcal{D}_u$, to extract its final state,

and set $\mathcal{S}$ to be that state. By a simple induction, we can see that after we finished processing node u_i, the final state of $\mathcal{D}_{u_i}$ corresponds to the state resulting from the concatenation of $S_{u_1}, S_{u_2}, \ldots, S_{u_i}$. Therefore, we can then query $\mathcal{D}_{u_k}$ to get the answer we want. Finally, we delete all the operations we added in those $\mathcal{D}_u$, so they can be used for the future queries. To summarize, we invoke partially retroactive update/query $O(n \cdot \log m)$ times, and hence the whole query takes $O(n \cdot \log m \cdot T_{\mathsf{op}}(n, m))$ time.

Demaine et al. [7] showed a reduction with $O(\sqrt{m})$ overhead. Roughly, their transformation maintains $\sqrt{m}$ equally distributed checkpoints, and for each checkpoint, they maintain a partially retroactive data structure for the prefix up to that checkpoint. For update, they need to update all the $\sqrt{m}$ partially retroactive data structures; for query of a prefix, they first find the closest checkpoint, adding or deleting operations to this checkpoint in order for it to match the prefix, and then do the query. For both update and query, there are $O(\sqrt{m})$ calls to the partially retroactive data structure, hence the $O(\sqrt{m})$ overhead.

Combining these two transformations gives an $O(\min\{\sqrt{m}, n \cdot \log m\})$ overhead. There is a subtle issue here as this requires us to know n and m beforehand. We can avoid that by using the standard technique that maintains two structures $\mathcal{D}_1$ and $\mathcal{D}_2$ simultaneously, one with $\sqrt{m}$ overhead and one with $n \cdot \log m$ overhead. We simulate $\mathcal{D}_1$ and $\mathcal{D}_2$ in an interleaving fashion, and answer the query as soon as one of them gives its answer. ◀

4 Discussion

Many lower bounds for algorithm problems are based on plausible conjectures from fine-grained complexity theory. Besides the three canonical ones (SETH, APSP, 3-SUM) mentioned above, some interesting hardness candidates include Boolean Matrix Multiplication [27], Online Matrix Vector Multiplication [19], and the Triangle Collection problem [2]. Their relationship and applications are discussed in detail in [26].

Our lower bound constructions reveal that fully retroactive queries facilitate batched pair evaluation. We believe this technique can prove useful for other data structure lower bounds, especially dynamic ones. Some examples include the total update time for partially-dynamic algorithms, worst-case update time, query/update time tradeoffs [19], and space/time tradeoffs [16].

References

1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science*, pages 477–486, 2016.

2 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 41–50, New York, NY, USA, 2015. ACM.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443, 2014.

4 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2007.

5 Guy E. Blelloch. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 894–903, 2008.

6 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median,+)-convolution, and some geometric 3sum-hard problems. In *Proceedings of SODA 2018*, 2018. to appear.

7 Erik D. Demaine, John Iacono, and Stefan Langerman. Retroactive data structures. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 274–283, 2004.

8 Erik D. Demaine, John Iacono, and Stefan Langerman. Retroactive data structures. *ACM Transactions on Algorithms*, 3(2):13:1–13:21, 2007.

9 Erik D. Demaine, Tim Kaler, Quanquan Liu, Aaron Sidford, and Adam Yedidia. Polylogarithmic fully retroactive priority queues via hierarchical checkpointing. In *Proceedings of the 14th International Symposium on Workshop on Algorithms and Data Structures*, pages 263–275. Springer, 2015.

10 Matthew T. Dickerson, David Eppstein, and Michael T. Goodrich. Cloning voronoi diagrams via retroactive data structures. In *Proceedings of the 18th Annual European Symposium on Algorithms*, pages 362–373, 2010.

11 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

12 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971*, pages 129–131, 1971.

13 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5:165–185, 1995.

14 Igal Galperin and Ronald L. Rivest. Scapegoat trees. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1993.

15 Yoav Giora and Haim Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Trans. Algorithms*, 5(3):28:1–28:51, 2009.

16 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 421–436, Cham, 2017. Springer International Publishing.

17 Michael T. Goodrich and Joseph A. Simons. Fully retroactive approximate range and nearest neighbor searching. In *Proceedings of the 22nd International Symposium on Algorithms and Computation*, pages 292–301, 2011.

18 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Proceedings of the IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 621–630, 2014.

19 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 21–30, 2015.

20 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. *arXiv preprint arXiv:1703.01638*, 2017.

21 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1272–1287, 2016.

23 Yakov Nekrich. Searching in dynamic catalogs on a tree. *CoRR*, abs/1007.3415, 2010. `arXiv:1007.3415`.

24 Salman Parsa. *Algorithms for the Reeb Graph and Related Concepts*. PhD thesis, Duke University, 2014.

25 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610, 2010.

26 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018. To appear.

27 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 645–654, Washington, DC, USA, 2010. IEEE Computer Society.